МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки, специальности)

Разработка программного обеспечения
(направленность (профиль)/специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка веб-приложения учёта финансов»

Обучающийся А. Д. Чистов
(И.О. Фамилия) (личная подпись)

Руководитель к.т.н, Н. В. Хрипунов
(ученая степень, звание, И.О. Фамилия)

Консультант старший преподаватель, И. Ю. Усатова
(ученая степень, звание, И.О. Фамилия)

Тольятти 2025

Аннотация

Данная выпускная квалификационная работа посвящена разработке веб-приложения учёта финансов для ПАО «Росбанк», направленного на достижение ключевых бизнес-целей: повышение точности и эффективности финансового учета (сокращение времени на планирование бюджета на 30%), снижение риска операционных ошибок (на 15%) и увеличение прозрачности финансовых данных (на 15% согласно внутренним KPI).

Объектом исследования бакалаврской работы является процесс составления бюджета (планирование и анализ) в ПАО «Росбанк».

Предметом исследования является функциональность веб-приложения, автоматизирующая планирование и анализ бюджетов, а также ее влияние на скорость, точность и трудозатраты.

Актуальность темы обусловлена необходимостью автоматизации и оптимизации финансовых процессов в банке для повышения конкурентоспособности и соблюдения нормативных требований ЦБ РФ.

Целью работы является создание единой платформы для анализа и планирования финансовых операций, обеспечивающей централизованное хранение данных о транзакциях различных банковских отделений, чёткую и понятную визуализацию данных, детализацию транзакций, а также повышение прозрачности и контроля финансовых потоков.

Для достижения цели были поставлены и решены следующие задачи: анализ существующих процессов финансового учета в банке, определение требований к функциональности приложения, проектирование архитектуры и базы данных (PostgreSQL), разработка пользовательского интерфейса на Javascript-фреймворке Angular и серверной части приложения на Java-фреймворке Spring Boot, тестирование и отладка приложения.

Работа состоит из введения, трех глав, заключения и списка использованных источников. Общий объем работы составляет 69 страниц, из которых 43 рисунка, 5 таблиц и 20 источников информации.

Abstract

The title of the graduation work is Development of a Web-based Financial Accounting Application.

The graduation work consists of an introduction, three parts, 43 figures, 5 tables, a conclusion, and a list of 20 references including foreign sources.

The aim of this graduation work is to develop the financial accounting web application designed for budget planning and analysis in the banking organization.

The object of the graduation work is the financial accounting in the banking organization.

The subject of the graduation work is the functionality of the web application that automates budget planning and analysis.

The key issue of the graduation work is to reduce the amount of manual work and errors in budget planning and analysis in the banking organization.

The first part describes in details the analysis of the subject area, including the business processes of the banking department, the requirements for the web application, the analysis of ready-made solutions and the software solution. We examine the processes of remote banking services and also choosing the appropriate system architecture.

The second part outlines the results of the analysis, which include the choice of the design methodology, the development of a use case diagram, domain data modeling, and web interface design.

The third part consists of developing a component diagram, selecting software tools, creating a database, and developing and testing a web-based financial accounting application.

In conclusion we'd like to stress based on the results of this work, a web application has been developed that automates financial accounting (budget planning and analysis) in a banking organization. Nevertheless, more experimental data are required.

Содержание

Введение

В современном мире эффективное управление финансами является ключевым фактором экономической стабильности как для отдельных граждан, так и для организаций. В условиях постоянно меняющейся экономической ситуации и роста числа финансовых инструментов, автоматизация учета финансов становится необходимостью. Особенно актуальна данная задача для банков, которые оперируют огромными объемами финансовых данных и нуждаются в надежных и эффективных инструментах для их обработки и анализа.

В настоящее время существует множество программных продуктов для учета финансов, как специализированных банковских систем, так и универсальных бухгалтерских программ. Однако, многие из них либо являются дорогостоящими и сложными в освоении, либо не обладают достаточной функциональностью для удовлетворения специфических потребностей банка. Кроме того, часто отсутствует полная интеграция между различными системами учета, что приводит к дублированию данных и снижению эффективности работы.

В связи с этим, разработка специализированного веб-приложения для учета финансов, адаптированного к потребностям конкретного банка, является актуальной задачей, позволяющей повысить эффективность работы, снизить риски и обеспечить прозрачность финансовых данных.

Данное веб-приложение позволит собирать и анализировать финансовые данные в режиме реального времени, что улучшит точность прогнозов доходов и расходов. Визуализация финансовых данных в виде графиков и диаграмм облегчит их восприятие и анализ. В целом, автоматизация процессов планирования и бюджетирования снизит трудозатраты и время, необходимое для подготовки бюджетов.

Целью данной выпускной квалификационной работы является разработка веб-приложения учёта финансов, предназначенного для

использования в ПАО «Росбанк». Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ предметной области и определить требования к функциональности веб-приложения;
- разработать архитектуру веб-приложения и спроектировать базу данных для хранения финансовых данных;
- реализовать пользовательский интерфейс веб-приложения, обеспечивающий удобный доступ к функциям учета, анализа и планирования финансов;
- разработать серверную часть веб-приложения, обеспечивающую обработку данных и взаимодействие с базой данных;
- провести тестирование и отладку веб-приложения.

1 Анализ предметной области

1.1 Характеристика предприятия – объекта исследования

Объектом исследования ВКР являются процессы дистанционного банковского обслуживания (ДБО) публичного акционерного общества «Росбанк». Данный банк является ведущим коммерческим банком России, он предоставляет широкий спектр финансовых услуг для физических и юридических лиц.

Основная цель коммерческого банка – получение прибыли от предоставления финансовых услуг. Однако это не просто учреждение, которое зарабатывает деньги, он играет важную роль в развитии экономики и социальной сферы. Всё это достигается путем:

- привлечения средств от клиентов: банк привлекает деньги от физических и юридических лиц в виде депозитов, кредитов, инвестиций;
- предоставления кредитов и других финансовых услуг: банк предоставляет деньги в кредит физическим и юридическим лицам, а также оказывает другие финансовые услуги (инвестирование, страхование, банковские гарантии);
- разница между процентными ставками: банк зарабатывает деньги на разнице между процентными ставками, которые он платит по депозитам, и процентными ставками, которые он получает по кредитам;
- инвестирование средств: банк инвестирует денежные ресурсы в различные проекты, которые принесут прибыль: акции, облигации, золото, нефть или сельхозпродукцию, недвижимость [12].

Для того, чтобы достичь своей цели, коммерческий банк решает следующие задачи:

- предоставление финансовых услуг,

– привлечение финансовых ресурсов,
– управление финансовыми рисками,
– развитие технологий и инноваций,
– обеспечение соответствия законодательству,
– участие в социальных проектах и инициативах [13].

В современном цифровом мире ДБО стало популярным инструментом для облегчения финансовых процессов, как для банков, так и для их клиентов. На рисунке 1 рассмотрим организационную структуру Центра компетенций корпоративных систем ДБО.

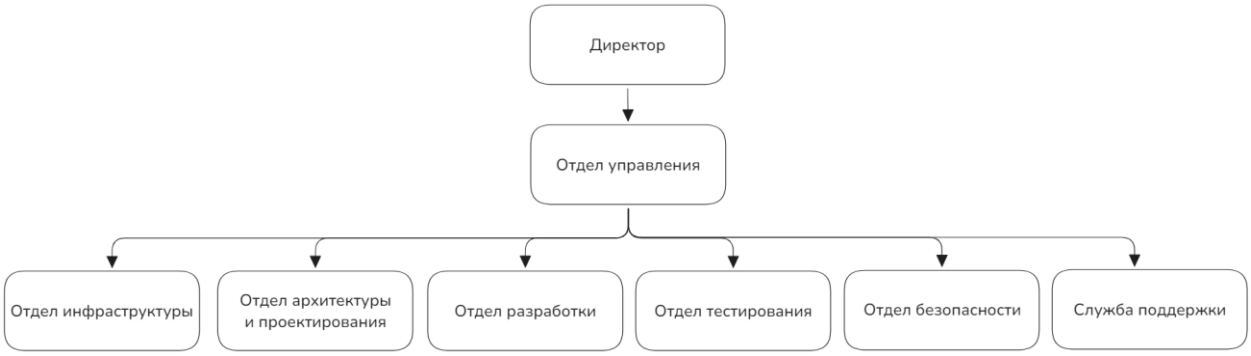


Рисунок 1 – Организационная структура департамента ДБО

Директор Центра отвечает за общее руководство центром, стратегическое планирование, управление бюджетом, контроль за выполнением задач, формирование стратегии развития. Он утверждает бюджеты проектов, принимает решения по приоритетам, назначает руководителей отделов, согласовывает стратегию развития. Взаимодействует с руководством банка, руководителями других подразделений и отделов.

Отдел управления руководит другими отделами, планирует и контролирует реализацию проектов, управляет рисками, взаимодействует с заказчиками, утверждает планы проектов, назначает руководителей отделов, согласовывает бюджеты, контролирует срок и качество проектов. Взаимодействует с директором Центра и руководителями отделов.

Отдел инфраструктуры обеспечивает работу серверов и сетей, разрабатывает и внедряет решения по масштабированию систем, управлению IT-инфраструктурой. Руководитель данного отдела утверждает решения по масштабированию, назначает ответственных за сервера и сети, согласовывает бюджет на IT-инфраструктуру. Отдел инфраструктуры взаимодействует с отделами управления, архитектуры и проектирования, разработки, тестирования и безопасности.

Отдел архитектуры и проектирования разрабатывает архитектуру системы ДБО, проектирует основной функционал, создаёт техническую спецификацию. Утверждает архитектурные решения, согласовывает технологические стеки, выбирает платформы и инструменты разработки. Взаимодействует с отделами инфраструктуры, разработки и безопасности.

Отдел разработки занимается проектированием и написанием кода системы ДБО, реализует функционал в соответствии с техническими спецификациями. Вносит предложения по оптимизации кода, выбирает инструменты разработки в рамках проекта. Взаимодействует с отделами управления, инфраструктуры, архитектуры и проектирования, тестирования.

Отдел тестирования проводит тестирование систем ДБО, выявляет ошибки и несоответствия требованиям. Имеет полномочия заблокировать выпуск нового функционала в производственную среду до устранения критических ошибок. Взаимодействует с отделами управления, инфраструктуры, разработки и безопасности.

Отдел безопасности обеспечивает безопасность систем ДБО, разрабатывает и внедряет политики информационной безопасности (ИБ), контролирует за соблюдением правил ИБ, управляет инцидентами. Руководитель отдела безопасности утверждает политики ИБ, назначает ответственных за безопасность, принимает решения по устранению уязвимостей. Данный отдел взаимодействует с управлением, отделами инфраструктуры, архитектуры и проектирования, тестирования и службой поддержки.

Служба поддержки занимается решением проблем пользователей, заводит инциденты, связанные с ошибками и неисправностями в системе ДБО. Вносит предложения по улучшению работы системы, выполняет задачи по настройке и обслуживанию системы. Взаимодействует с отделами управления и безопасности.

1.2 Описание бизнес-процессов рассматриваемого подразделения

Данные всех транзакций отдела ДБО отправляются не только в базу данных и CRM-систему, но также и в стороннюю систему учета финансов. Система учета финансов нужна для анализа финансовых потоков руководством отдела ДБО и аналитиками. В ПАО «Росбанк» множество отделов, каждый из которых использует свою систему учета финансов.

Построим функциональную модель «AS-IS» (рисунки 2-3) бизнес-процесса бюджетирования – это поможет понять, каким образом сотрудники отдела управления финансами агрегируют данные для их последующего анализа и планирования бюджета.

«IDEF0, ранее известная как технология структурированного анализа и разработки (SADT – Structured Analysis and Design Technique), является стандартом технологии моделирования процессов. Одной из основных идей IDEF0-моделей является построение двух видов моделей: «как есть» (AS-IS) и «как должно быть» (TO-BE). Это нужно при проведении реинжиниринга бизнес-процессов организации. При проведении сложных проектов обследования предприятий, разработка моделей в стандарте IDEF0 позволяет наглядно и эффективно отобразить весь механизм деятельности предприятия в нужном разрезе.» [9, с. 33].

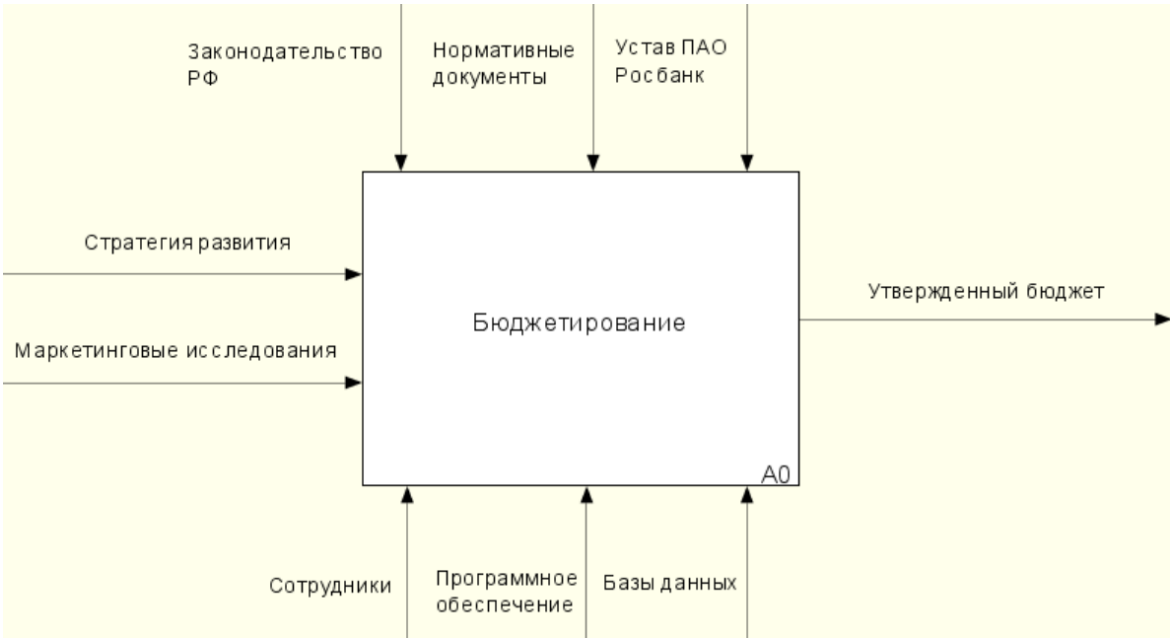


Рисунок 2 – Бюджетирование (нотация IDEF0, модель «AS-IS»)

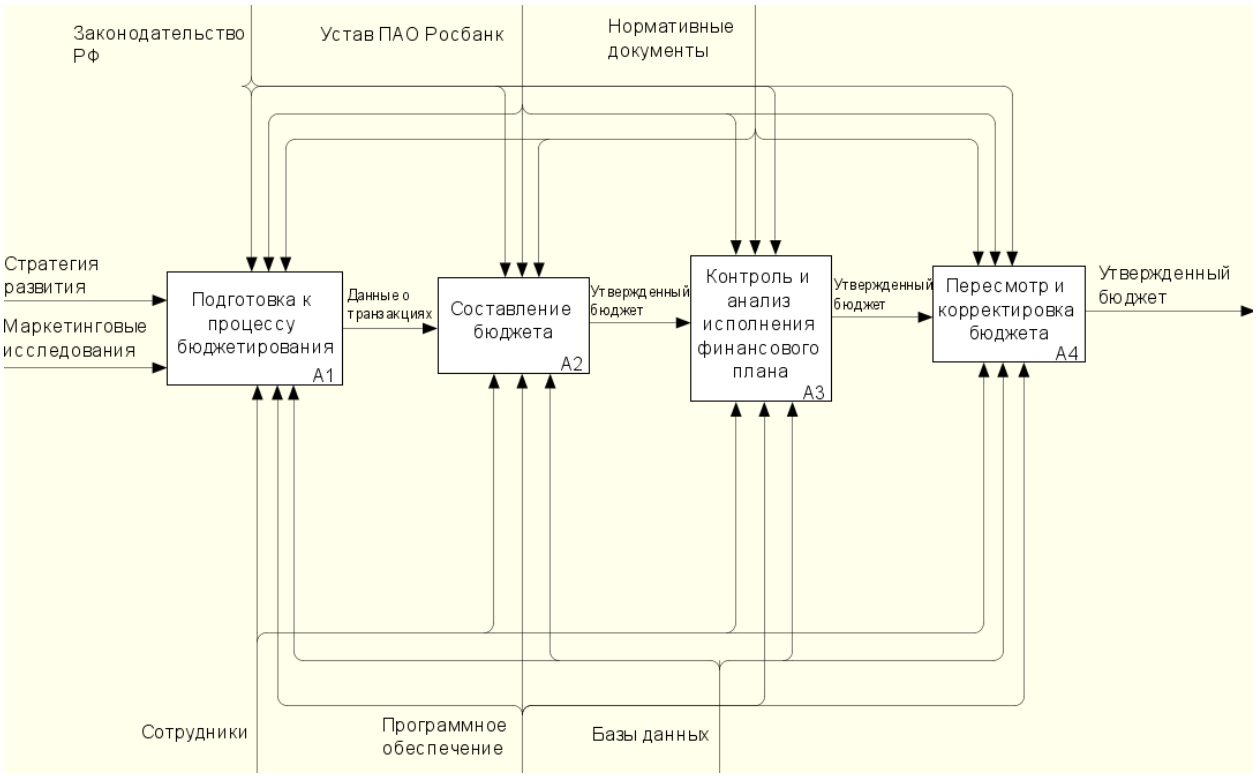


Рисунок 3 – Бюджетирование (нотация IDEF0, модель «AS-IS»)

Можно предположить, что агрегация финансовых данных для анализа и планирования бюджета происходит в подпроцессе «Составление бюджета».

На рисунке 4 рассмотрим функциональные процессы составления бюджета, связанные потоками данных в нотации DFD, это позволит рассмотреть процесс подробнее.

«Диаграммы потоков данных (Data Flow Diagrams - DFD) представляют собой иерархию функциональных процессов, связанных потоками данных. Цель такого представления – продемонстрировать, как каждый процесс преобразует свои входные данные в выходные, а также выявить отношения между этими процессами.» [5, с. 33].

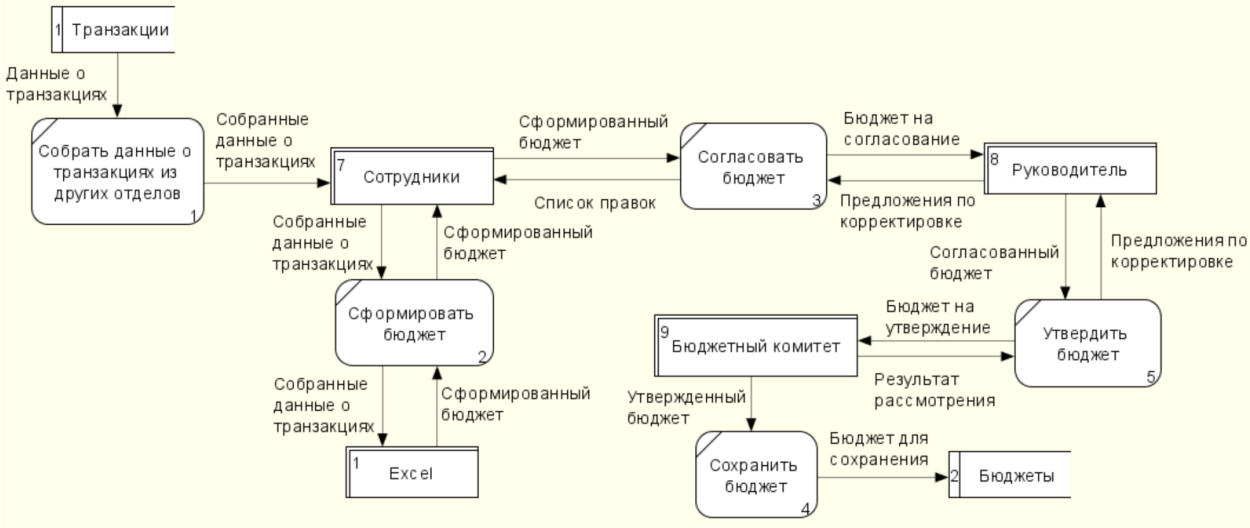


Рисунок 4 – Составление бюджета (нотация DFD, модель «AS-IS»)

Из диаграммы бизнес-процесса составления бюджета видно, что данные о транзакциях из разных отделов обрабатываются с помощью программы Excel. Сотрудники управления финансами импортируют данные в Excel из баз данных отделов, обрабатывают и анализируют их, формируют бюджет, который согласовывается с руководителем отдела и утверждается Бюджетным комитетом, затем сохраняется в базу данных для дальнейшего контроля и анализа исполнения финансового плана и его корректировки в будущем.

В результате проведенного анализа был выявлен бизнес-процесс, требующий автоматизации, основная его проблема – отсутствие единой системы учета финансов, вследствие этого затрудняется сбор данных, анализ и принятие эффективных управленческих решений.

1.3 Анализ лучших практик в предметной области

В современной банковской сфере составление бюджета выходит за рамки простой таблицы в Excel. Используются комплексные подходы, интегрированные инструменты и специализированное ПО, благодаря этому повышается точность прогнозирования, эффективность управления финансами и соответствие регуляторным требованиям.

С целью обеспечения эффективного и точного финансового учета в банковской сфере, необходимо придерживаться лучших практик, рассмотрим их ниже:

- современные банки активно интегрируют данные из разных систем: CRM, системы кредитования, инвестирования, депозитов, операций с ценными бумагами, и другие;
- API (Application Programming Interface) позволяет обмениваться данными между разными системами без необходимости ручного ввода или экспорта-импорта данных;
- использование инструментов BI (Business Intelligence) для анализа финансовых показателей, выявления тенденций и создания отчетов;
- применение ИИ и машинного обучения для прогнозирования финансовых показателей.

Предложенное решение проблемы заключается в разработке веб-приложения учёта финансов, которое будет агрегировать данные из различных информационных систем отделов банка. Данная система позволит:

– улучшить аналитику: агрегация данных из разных отделов позволит проводить более глубокий анализ финансовых показателей и выявлять тенденции;
– сократить время на сбор данных: автоматизация сбора и обработки данных из разных отделов сократит время на подготовку отчетов и аналитических материалов;
– повысить точность отчетности: единая система учета финансов снизит риск ошибок и повысит точность отчетности.

Рассмотрим следующие преимущества веб-приложения по сравнению с другими видами программного обеспечения:

– веб-приложение доступно с любого устройства с доступом в интернет;
– обновления программного обеспечения автоматически доступны всем пользователям без необходимости загрузки новых версий;
– отсутствие необходимости получать разрешение от сторонней платформы, как в случае с мобильными приложениями, для выхода на рынок;
– веб-приложение можно легко масштабировать в соответствии с ростом банка и количества пользователей;
– современные веб-технологии позволяют обеспечить высокий уровень безопасности данных.

1.4 Системная архитектура проекта

При выборе архитектуры были рассмотрены следующие важнейшие принципы проектирования ИС:

– разделение ответственности: архитектура должна разделять приложение на логические уровни, например: клиентский (веб-интерфейс), серверный (бизнес-логика и API) и уровень базы данных (хранилище данных). Такое разделение упрощает

разработку, тестирование, поддержку и масштабирование приложения. Каждый уровень может разрабатываться и изменяться независимо от других, что снижает сложность и риски;

– безопасность: архитектура должна позволять более эффективно обеспечивать безопасность данных, например если бизнес-логика и доступ к базе данных сосредоточены на серверном уровне, это позволит реализовывать более строгие механизмы контроля доступа и шифрования данных;
– целостность данных: выбранная архитектура должна обеспечивать централизованное хранение данных, такой подход обеспечит целостность данных, одновременный доступ многих пользователей и возможность проведения сложного анализа данных.

Исходя из описанных выше критериев, была выбрана трехуровневая (трехзвенная) клиент-серверная архитектура. Она не только соответствует описанным выше принципам проектирования ИС, но также решает проблемы, связанные с использованием Excel для финансового учета: трудности в совместной работе, риск потери данных, ограничения в функциональности.

В основе функционирования веб-ресурсов лежит протокол HTTP, который организует взаимодействие между клиентом и сервером. При этом:

– клиент (например, ваш браузер) инициирует и отправляет запрос к серверу (где размещены данные веб-ресурса);
– сервер обрабатывает этот запрос и отправляет ответ обратно клиенту, который отображает полученную информацию.

Клиенты бывают двух типов:

– тонкий клиент: основную работу по обработке информации выполняет сервер, а клиент только отображает результат. Пример – браузер, используемый для просмотра веб-страниц;

– толстый клиент: клиент сам обрабатывает большую часть информации, а сервер использует в основном для хранения данных [11].

Идея «тонкого клиента» предполагает разделение алгоритмов обработки данных на два ключевых компонента: реализацию бизнес-функций и представление информации пользователю. В рамках данной концепции, клиентский компьютер выполняет минимальный набор задач, ограничиваясь отображением данных и первичной проверкой вводимой пользователем информации. Основная вычислительная нагрузка, связанная с выполнением бизнес-логики, размещается на сервере, что способствует повышению эффективности и упрощению управления системой. Данная архитектурная модель, известная как трехзвенная архитектура, схематично изображена на рисунке 5 [1].

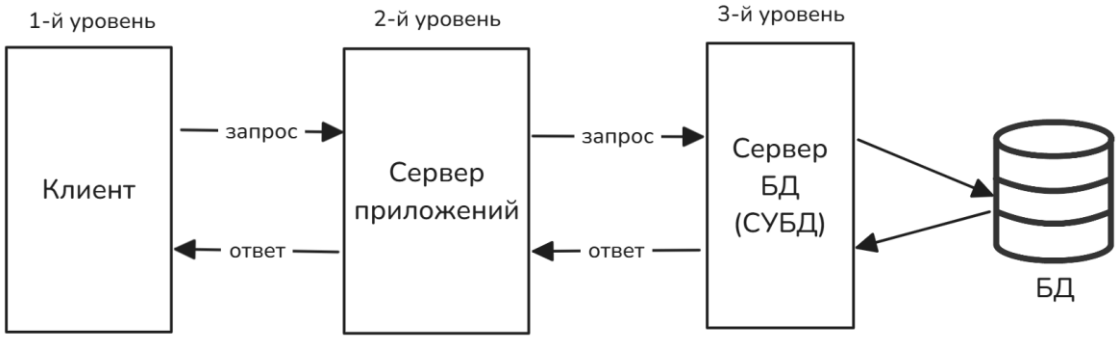


Рисунок 5 – Трехзвенная архитектура

Так как мы разрабатываем веб-приложение, в данном случае веб-браузер выполняет всю логику представления, он предоставляет пользователям интуитивный интерфейс для работы с системой. Этот уровень отвечает за отображение данных, обработку пользовательского ввода и взаимодействие с серверным уровнем через API.

Серверная часть приложения отвечает за обработку данных, взаимодействие с базой данных, аутентификацию пользователей и бизнес-

логику. Сервер будет обеспечивать RESTful API для взаимодействия с клиентским приложением.

На третьем уровне располагается система управления базами данных (СУБД), например реляционными (PostgreSQL, MySQL), для хранения информации о пользователях, бюджетах, финансовых операциях, и других данных. В крупных организациях для уровня баз данных используется выделенный сервер или кластер серверов.

1.5 Требования к веб-приложению

Изучив предметную область и представленные на рынке системы учета финансов, были сформулированы оптимальные требования к веб-приложению. Подробно опишем их в таблице 1 на основе модели FURPS+ – мнемоническая аббревиатура, обозначающая различные категории атрибутов качества программного обеспечения.

Таблица 1 – Классификация требований к разрабатываемой системе

Требование	Статус	Полезность	Риск
Функциональные требования			
Пользователь должен иметь возможность входа в систему с использованием логина и пароля	Одобренные	Важное	Низкий
Система должна разграничивать роли и права доступа для разных пользователей (аналитики, менеджеры, руководители)	Одобренные	Критическое	Низкий
Пользователь с ролью администратора должен иметь возможность добавлять пользователей в систему, удалять и задавать им необходимую роль	Одобренные	Критическое	Низкий
Пользователь с ролью администратора или менеджера должен иметь возможность создавать, удалять и редактировать бюджеты	Одобренные	Критическое	Низкий
Пользователь с ролью администратора или менеджера должен иметь возможность загружать данные о бюджетах в систему из Excel файлов	Одобренные	Важное	Низкий

Продолжение таблицы 1

Требование	Статус	Полезность	Риск
Пользователь с ролью администратора или менеджера должен иметь возможность выгружать данные о бюджетах из системы в Excel файлы	Одобренные	Важное	Низкий
Пользователь с ролью администратора или менеджера должен иметь возможность добавлять денежные операции в систему и просматривать их	Одобренные	Критическое	Низкий
Система должна отображать графики и диаграммы для визуализации данных по отдельным бюджетам	Одобренные	Критическое	Низкий
Пользователь должен иметь возможность фильтровать данные по критериям (дата, отдел, тип операции)	Одобренные	Критическое	Низкий
Удобство использования			
Новый пользователь должен иметь возможность освоить основные функции системы за 1 час	Одобренные	Важное	Низкий
Система должна иметь подробную справочную информацию и руководство пользователя	Одобренные	Важное	Низкий
Система должна быть доступна с разных устройств (компьютеры, мобильные устройства).	Предложенные	Важное	Низкий
Надежность			
Система должна быть доступна 24/7 с гарантированной доступностью 99.8%	Одобренные	Критическое	Средний
Время восстановления системы после сбоя не должно превышать 15 минут	Одобренные	Важное	Средний
Система должна обеспечивать точность финансовых расчетов не менее 99.98%	Одобренные	Критическое	Высокий
Производительность			
Время отклика системы на запрос не должно превышать 2 секунды	Одобренные	Важное	Средний
Поддерживаемость			
Система должна иметь модульную архитектуру, облегчающую проведение тестирования	Одобренные	Важное	Средний
Внесение изменений в систему не должно требовать значительных усилий или не должно приводить к возникновению новых ошибок	Одобренные	Важное	Средний
Ограничения			
Система должна соответствовать требованиям законодательства о защите персональных данных (например, 152-ФЗ), к банковской тайне и другим нормативным актам	Одобренные	Критическое	Высокий

Продолжение таблицы 1

Требование	Статус	Полезность	Риск
Система должна соответствовать требованиям ЦБ РФ к ведению финансового учета и отчетности	Одобренные	Критическое	Высокий

Сформулированные являются основой для поиска возможности использовать готовое решение.

1.6 Анализ готовых решений

После проведения анализа существующих разработок для решения обозначенной задачи, можно выделить, что системы автоматизации бюджетирования разделяются по способу внедрения на интегрируемые и автономные. Рассмотрим их достоинства и недостатки в таблице 2.

Таблица 2 – Сравнительный анализ систем учета финансов и бюджетирования [8]

Критерий	Наименование				
	ИТАН: Управленческий Баланс	БИТ: ФИНАНС	WS: Финансист	Инталев: Корпоративный менеджмент	Хомнет: МСФО
Проектное внедрение (руб.)	от 3 000 000	от 4 000 000	2 950 000	от 10 000 000	от 700 000
Возможность интеграции в виде подсистемы	Да	Да	Нет	Да	Нет
Нормативное и коэффициентное планирование бюджета	Да	Да	Да	Да	Нет
Возможность ведения финансовых бюджетов (БДР, БДДС, Бюджет затрат)	Да	Да	Да	Да	Да

Продолжение таблицы 2

Критерий	Наименование				
	ИТАН: Управленческий Баланс	БИТ: ФИНАНС	WS: Финансист	Инталев: Корпоративный менеджмент	Хомнет: МСФО
Возможность загрузки данных по бюджету из Excel	Да	Да	Да	Да	Нет
Готовые механизмы загрузки данных из бухгалтерских систем учета на 1С 7.7 и 8	Да	Нет	Да	Нет	Да
Готовый API загрузки данных, в том числе через WEB сервис	Да	Нет	Да	Нет	Нет
Возможность вывода отчета в виде диаграмм	Да	Да	Да	Да	Нет
Возможность индивидуальной настройки разграничения прав для пользователей и объектов	Да	Да	Да	Да	Нет
Работа в Веб-клиенте	Частично	Да	Да	Да	Нет

Анализ готовых систем учета финансов выявил следующие их недостатки:

- избыточная функциональность,
- недостаточная адаптивность (программу сложно настраивать и внедрять),
- очень высокая стоимость некоторых из систем,
- некоторые из систем придется дорабатывать самостоятельно через разработку.

В банковской сфере есть особенности в соблюдении регуляторных требований (ЦБ, и других надзорных органов). В этом плане, гибкость (разработка собственного ПО) также является преимуществом. К тому же у рассматриваемого предприятия достаточно ресурсов для собственной разработки, которая вероятно даст конкурентное преимущество в будущем.

1.7 Описание программного решения

Итогом анализа была выбрана разработка веб-приложения учета финансов, которое будет агрегировать данные с возможностью удобной работы с ними и их представления в виде графиков и диаграмм. В данном случае, этот подход эффективнее, чем интеграция готового решения.

На основе вышеизложенного заключения построим графическое представление потока данных бизнес-процесса составления бюджета (модель «TO-BE») в нотации DFD (рисунок 6).

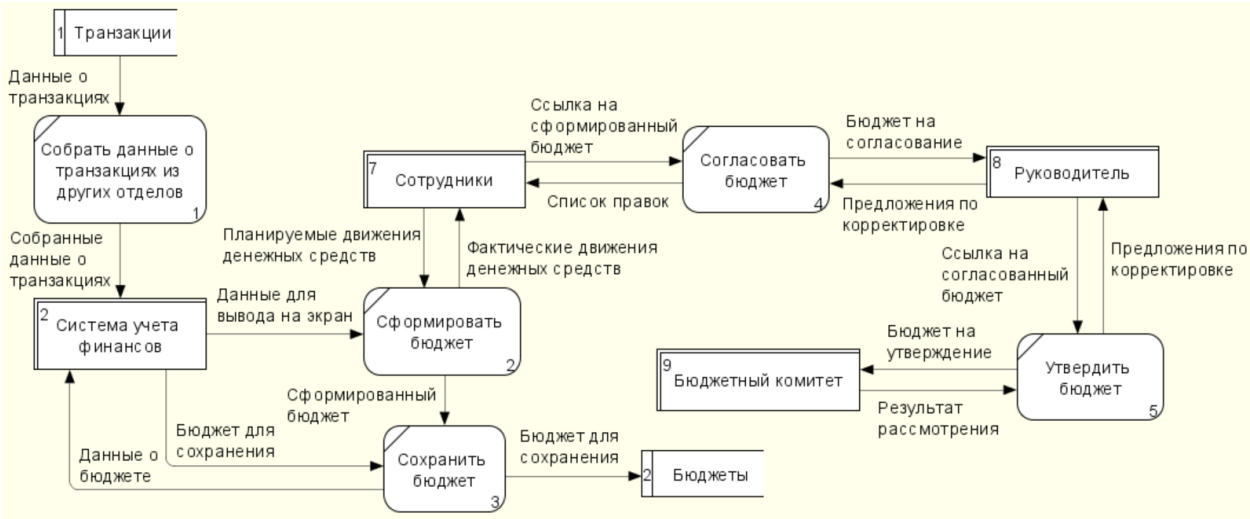


Рисунок 6 – Составление бюджета (нотация DFD, модель «TO-BE»)

В результате разработанная система учета финансов собирает данные из БД отделов, сохраняет в общей БД и представляет в удобном виде через веб-интерфейс. Специалисты финансового отдела смогут создавать бюджеты

в веб-приложении, а после согласования и утверждения бюджет также будет сохранён в общей базе данных и при необходимости будет отображаться в веб-приложении для работы с ним.

Вывод по разделу 1

В первом разделе ВКР был проведен всесторонний анализ предметной области, включающий изучение бизнес-процессов рассматриваемого банковского подразделения, лучших практик и готовых решений. На основе этого анализа определены оптимальные требования к веб-приложению и выбрана подходящая системная архитектура, что послужило надежным фундаментом для дальнейшей разработки программного решения, направленного на автоматизацию и оптимизацию выбранного бизнес-процесса.

2 Проектирование веб приложения учёта финансов

2.1 Выбор методологии проектирования

Выбор методологии проектирования программного обеспечения – важный шаг, определяющий подход к организации процесса разработки, управлению требованиями, коммуникациям в команде и общему успеху проекта. Не существует универсальной методологии, выбор зависит от множества факторов, связанных с проектом, командой и организацией.

Учитывая специфику банковской сферы (высокие требования к безопасности, надежности и соответствию нормативным требованиям), было принято решение использовать гибридный подход, сочетающий элементы традиционных и Agile-методологий:

- этап планирования и анализа требований: использовать элементы каскадной модели для четкого определения требований, разработки архитектуры и проектирования базы данных;
- этап разработки и тестирования: использовать Scrum для реализации отдельных компонентов веб-приложения, проведения итераций разработки, получения обратной связи от заказчика и адаптации к изменениям;
- этап внедрения и сопровождения: использовать элементы каскадной модели для контроля качества, обеспечения соответствия нормативным требованиям и планирования обновлений и улучшений [14].

2.2 Разработка диаграммы вариантов использования

Для того, чтобы уточнить функциональность системы и определить как пользователи будут с ней взаимодействовать спроектируем диаграмму прецедентов (вариантов использования) в UML.

Можно выделить такие цели создания диаграмм прецедентов:

– определение границы и контекста моделируемой предметной области на ранних этапах проектирования;
– формирование общих требований к поведению проектируемой системы;
– разработка концептуальной модели системы для ее последующей детализации;
– подготовка документации для взаимодействия с заказчиками и пользователями системы [2, с. 31].

На рисунке 7 представлено описание взаимоотношений и зависимостей между группами вариантов использования и участниками процессов.

Рисунок 7 – Диаграмма вариантов использования разрабатываемого веб-приложения

Данная диаграмма демонстрирует разделение ролей и прав доступа пользователей в системе, а также основные функциональные возможности системы бюджетного планирования:

- админ: пользователь с наивысшими правами доступа;
- менеджер: пользователь с правами управления бюджетом;
- аналитик: пользователь с правами просмотра отчетов.

2.3 Моделирование данных предметной области

На начальном этапе разработки нового приложения возникает необходимость четкого понимания структуры данных, с которыми предстоит работать системе. Для решения этой задачи используется моделирование данных предметной области, которое позволяет:

- обеспечить более эффективное взаимодействие между членами команды разработчиков;
- снизить вероятность ошибок при проектировании программного обеспечения и баз данных;
- определить информационную модель приложения, которая послужит основой для проектирования базы данных, и понять, какую информацию будет хранить и обрабатывать приложение;
- стандартизировать документацию компании, приведя ее к единому формату [10].

Начнём построение модели с концептуального уровня, цель этого уровня – понять суть предметной области, выделить основные сущности и связи. ER-диаграмма представляет собой графическое описание ER-модели предметной области системы. Для построения ER-диаграмм используются разные нотации. Нотация Чена, часто используемая для описания данных на

концептуальном уровне, обладает достаточным набором элементов, чтобы впоследствии перейти к логическому уровню моделирования.

На рисунке 8 представлена концептуальная схема ER-модели данных в нотации Чена разрабатываемого веб-приложения.

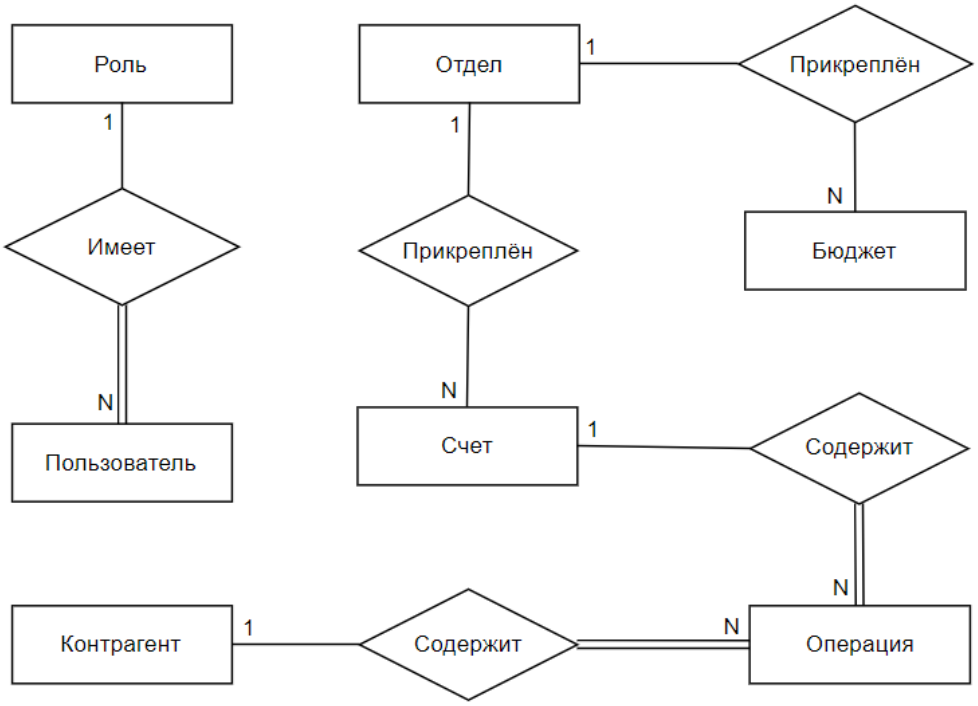


Рисунок 8 – Концептуальная схема данных разрабатываемого приложения

Пользователь обязательно имеет одну роль. К отделу могут быть прикреплены счета, но есть счета, которые могут не быть прикреплены и быть обще-банковскими. Сохраненный бюджет может прикрепляться к одному из отделов либо быть обще-банковским. Любая операция обязательно прикрепляется к какому-либо счёту и содержит контрагента. Контрагент – независимая сущность, и может быть незадействованной ни в одной операции.

Следующим уровнем моделирования данных является – логический. Логическая модель необходима для описания атрибутов сущностей, определения первичных ключей и уточнения числа сущностей – всё это

подготовительные действия перед составлением физической модели, которая ляжет в основу будущей базы данных (рисунок 9).

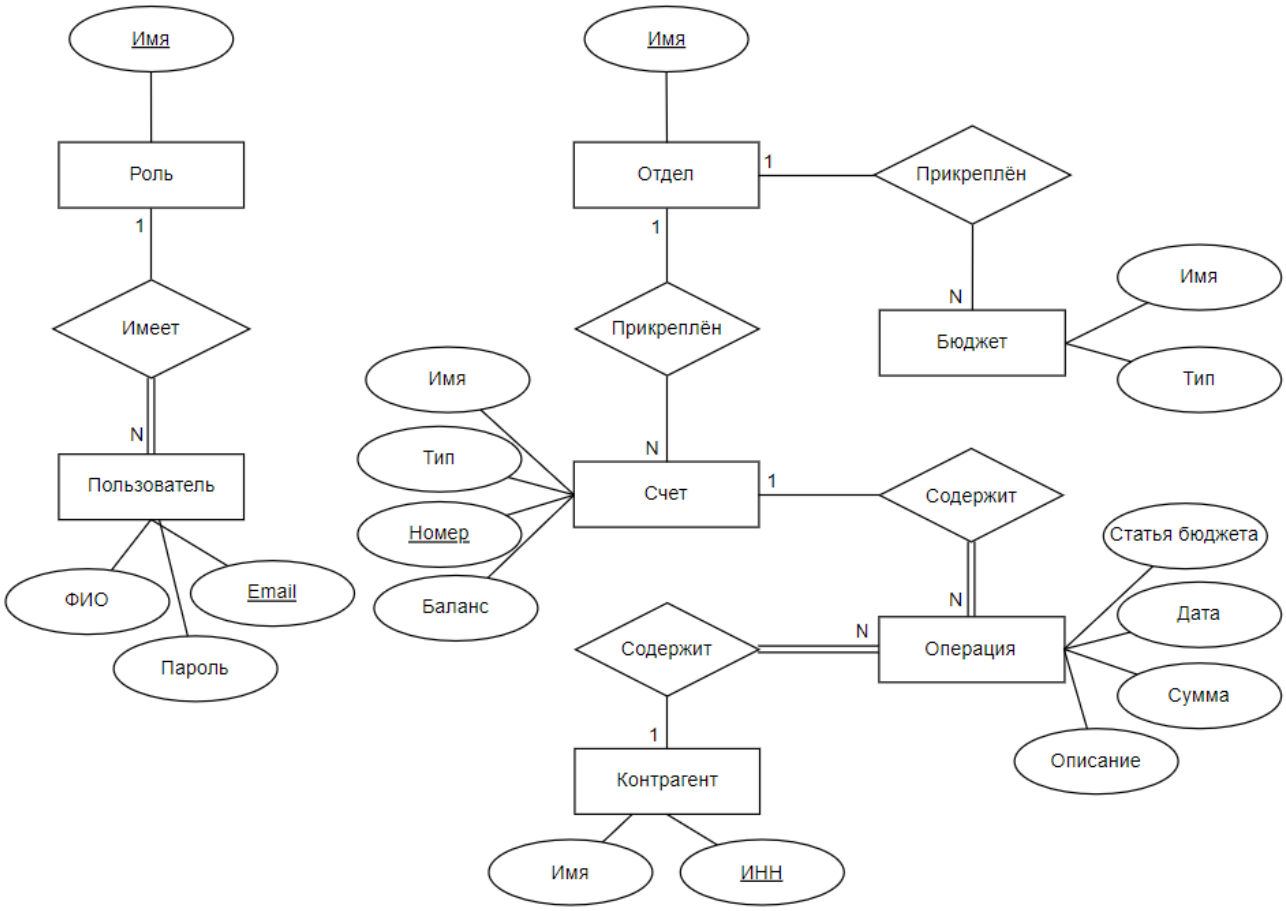


Рисунок 9 – Логическая схема данных разрабатываемого веб-приложения

Перед переходом к проектированию физической модели данных необходимо осуществить выбор типа хранилища данных. Для обеспечения эффективной работы приложения ключевым фактором является обеспечение быстродействия и отказоустойчивости выбранного хранилища. В настоящее время существует множество типов баз данных, включая реляционные, нереляционные, объектные и многомерные.

Реляционные базы данных представляют собой наиболее распространенный тип баз данных для приложений. Данные в них организованы в виде таблиц, связанных между собой. Примерами систем

управления реляционными базами данных (СУБД) являются Microsoft Access, Oracle, Microsoft SQL Server, MySQL и PostgreSQL.

В отличие от реляционных, данные в нереляционных БД (NoSQL) могут храниться в различных форматах, таких как документы (JSON), пары «ключ-значение» или графы. Основные типы нереляционных баз данных включают в себя:

- хранилища документов позволяют хранить документы с произвольным набором атрибутов и их значением, что делает их подходящими для проектов с часто меняющейся структурой данных;
- столбчатые хранилища данных организуют данные в таблицах, но при этом столбцы объединяются в семейства, что обеспечивает эффективное хранение и обработку больших данных;
- базы данных «ключ-значение» хранят данные в виде пар «ключ-значение», но не предоставляют доступ к внутреннему содержанию значений, поэтому требуется полная перезапись значений, что делает их подходящими для хранения изображений и файлов;
- графовые базы данных представляют данные в виде графов, где узлы представляют сущности, а ребра – связи между ними, что делает их оптимальными для приложений с большим количеством связей, таких как социальные сети.

Многомерные базы данных представляют собой тип хранилища данных, в котором информация организуется в виде упорядоченных многомерных массивов. В таких базах агрегированные данные хранятся непосредственно в массивах, а для их представления и анализа используются витрины данных. Многомерные базы данных часто применяются в системах, работающих с большими объемами информации (Big Data), где необходима многомерная аналитика.

Объектно-ориентированные базы данных, в свою очередь, организуют данные в соответствии с объектно-ориентированным подходом. В рамках

этой парадигмы, все сущности рассматриваются как объекты, обладающие как данными (атрибутами), так и методами (поведением), что позволяет моделировать более сложные иерархии и взаимосвязи между данными [6].

Реляционные базы данных являются проверенным и надежным решением для банков, обеспечивая структурированность, целостность, надежность и возможность работать со сложными отношениями между данными. Они соответствуют строгим требованиям банковской отрасли, что делает их основным выбором для большинства банковских систем.

Давайте посмотрим в таблице 3 на ключевые особенности самых популярных баз данных MySQL и PostgreSQL.

Таблица 3 – Сравнительный анализ популярных реляционных СУБД [17]

Параметры сравнения	СУБД	
	MySQL	PostgreSQL
Поддерживаемые языки	C/C++, Delphi, Erlang, Go, Java, Lisp, Node.js, Perl, PHP, R	C/C++, Delphi, Erlang, Go, Java, JavaScript, Lisp, .Net, Python, R, Tcl и другие
Чувствительность к регистру	Не различает регистр (любой регистр)	Различает регистр (требуется точное совпадение)
Преобразование наборов символов	Требуется UTF-8	Нет необходимости
Скорость	Высокая скорость при высокопараллельных операциях без записи в базе данных (только чтение)	Выигрывает в производительности при сложных операциях чтения-записи с одновременной валидацией данных
Надежность	Из-за формата таблиц MyISAM в InnoDB на MySQL могут возникнуть проблемы с поврежденными таблицами	База данных PostgreSQL предотвращает повреждение данных и сохраняет их целостность на транзакционном уровне

И MySQL, и PostgreSQL считаются одними из самых популярных, удобных и быстрых СУБД. Однако, в банковской сфере важна надежность,

поэтому выбор сделан в пользу PostgreSQL. Данный выбор основан не только на технических характеристиках, но и на практических соображениях, связанных с имеющимися у компании ресурсами и опытом. Отдельные БД, используемые для хранения бюджетов различных департаментов организации, также используют PostgreSQL. Таким образом, мы избегаем лишних затрат на обучение, консалтинг и можем сфокусироваться на разработке нового банковского продукта.

Теперь мы можем перейти от логической ER-модели к физической. Цель физического уровня ER-модели – описать структуру будущей базы данных.

В физическую модель данных (рисунок 10) была добавлена таблица budget_item, в ней будут хранится названия статей бюджетов и их иерархия. Иерархия статей бюджетов будет поддерживаться с помощью ключа bitem_parent_id, который будет указывать на id верхнеуровневой статьи бюджета. Если bitem_parent_id равен null, значит данный элемент является корневым.

Также в модель данных была добавлена таблица plan, в ней будут хранится суммы по месяцам. Каждый месяц представляет собой строку. В атрибуте plan_month_number будет хранится номер месяца (от 1 до 12).

В таблицу Plan было добавлено ограничение на колонку plan_month_number – она может принимать значения в диапазоне от 1 до 12. А в таблице department пара колонок (id и name) должны быть уникальными у каждой строки.

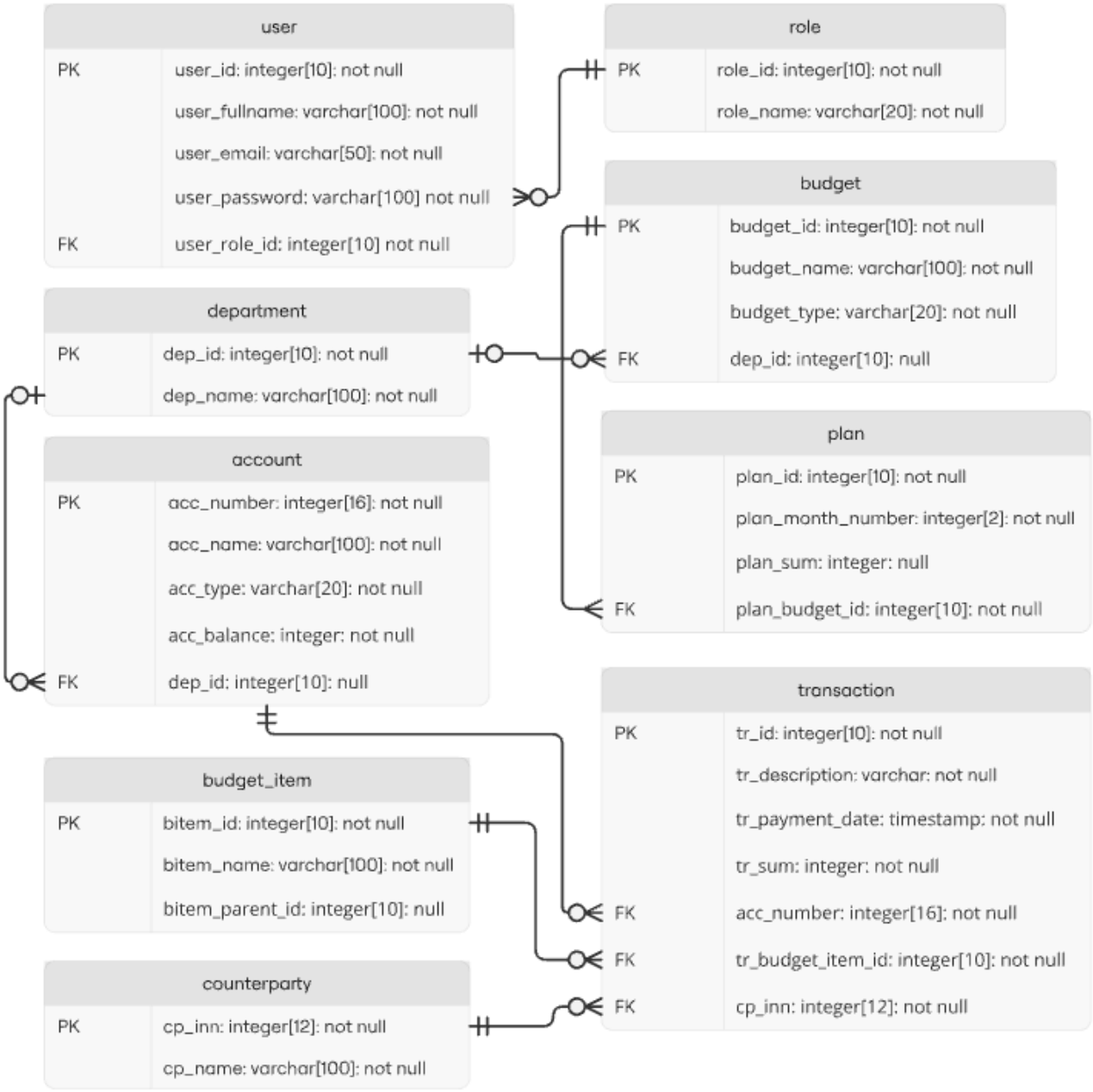


Рисунок 10 – Физическая схема данных разрабатываемого веб-приложения

Банковская система представляет собой сложную сеть взаимосвязанных объектов, включая клиентов, счета, транзакции и различные продукты. Объектно-ориентированный подход, основанный на классах и объектах, является оптимальным выбором для моделирования этих сложных взаимодействий. Классы, выступающие в роли основных строительных блоков объектно-ориентированной системы, описывают общие атрибуты, поведение и отношения между объектами. Для наглядного

представления структуры системы и ее компонентов используется диаграмма классов, которая упрощает понимание и облегчает процесс проектирования.

Диаграмма классов (рисунок 11) – это один из видов диаграмм UML, который показывает, как устроены классы в программе.

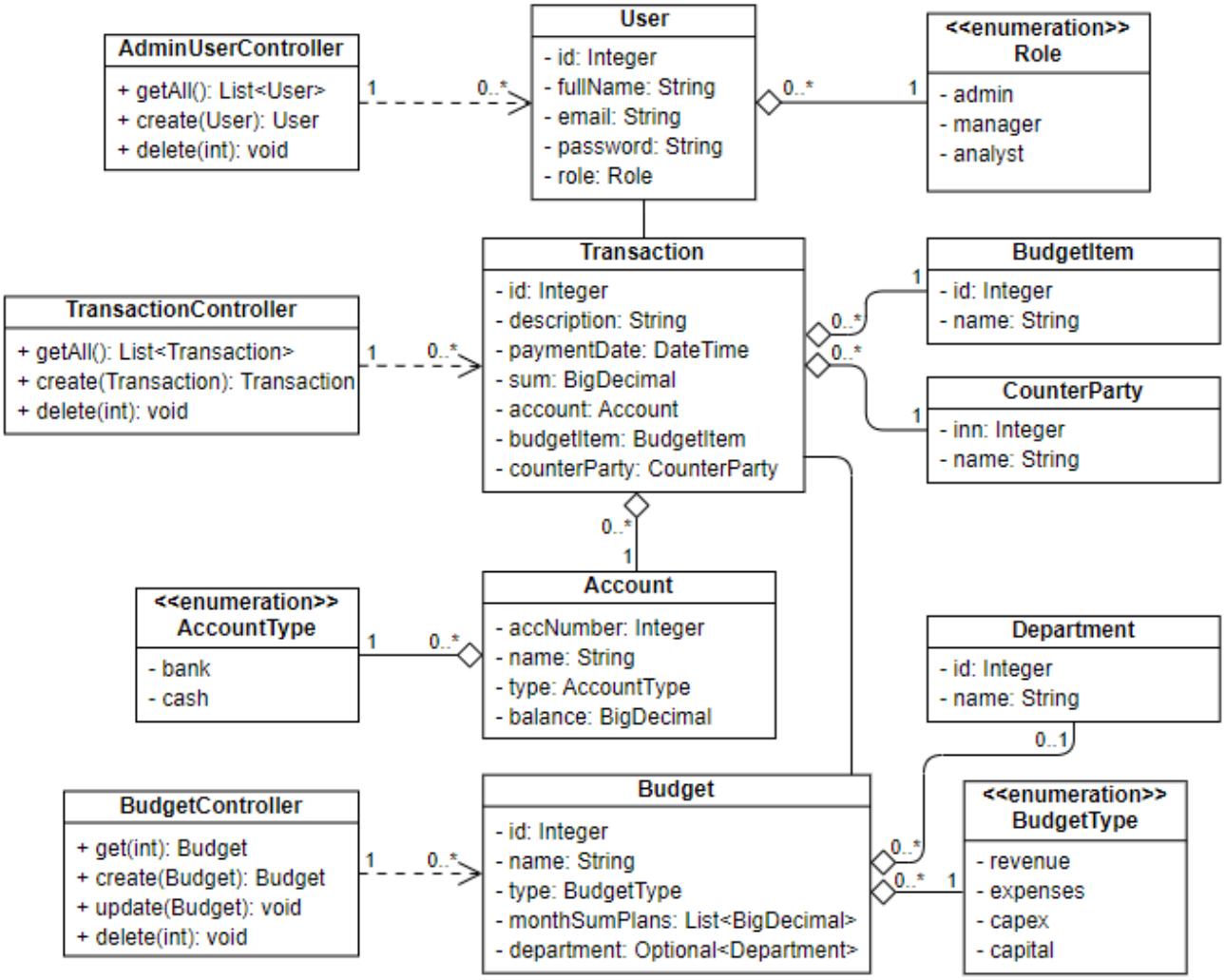


Рисунок 11 – Диаграмма классов разрабатываемого веб-приложения

На ней изображены классы с их свойствами (атрибутами) и методами (операциями), а также связи между этими классами. Эта диаграмма показывает структуру системы, как она выглядит на статичном уровне в объектно-ориентированном программировании. Помимо классов, на диаграмме могут быть показаны интерфейсы, пакеты и отношения.

Информация с диаграммы классов используется для создания исходного кода приложения. Многие инструменты для моделирования позволяют автоматически генерировать код, например на языках Java или C++. Таким образом, диаграмма классов является итогом проектирования и началом разработки приложения [7].

2.4 Проектирование веб-интерфейса

Чтобы интерфейс был функциональным и учитывал все потребности пользователей, нужно заранее спроектировать его структуру, то есть понять, из каких экранов будет состоять интерфейс. В этом поможет диаграмма структуры интерфейса.

Диаграмма структуры интерфейса (Interface Structure Diagram, ISD) – это визуальное представление, которое иллюстрирует взаимодействия и связи между системными интерфейсами. Диаграммы такого типа необходимы для отображения структуры сложных систем и лучшего понимания того, как различные интерфейсы взаимодействуют друг с другом. Они помогают визуализировать точки интеграции и зависимости внутри системы, что упрощает выявление потенциальных проблем и оптимизацию общего дизайна [16].

Каждый экран связан с другими, линиями. Они показывают, как пользователи могут переходить от одного экрана к другому. В большинстве случаев интерфейсы образуют иерархию или «дерево», которое ветвится вниз. Представим диаграмму структуры интерфейса разрабатываемого веб-приложения на рисунке 12.

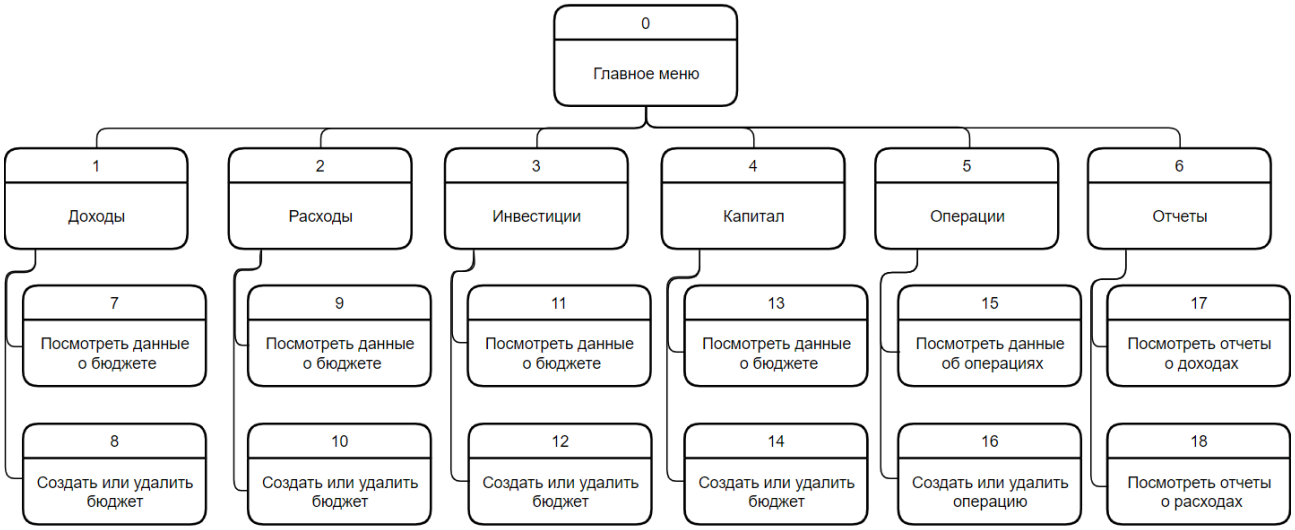


Рисунок 12 – Диаграмма структуры разрабатываемого веб-интерфейса

На рисунке 13, представим прототип страницы «Операции» в веб-программе Figma.

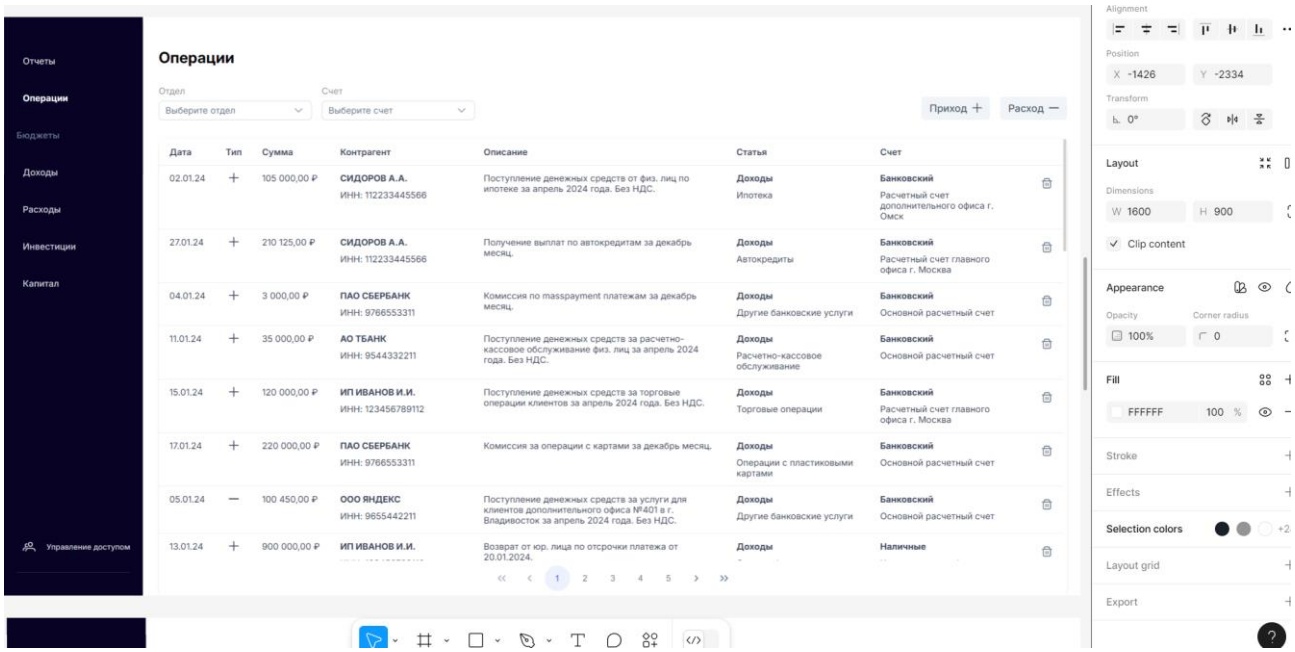


Рисунок 13 – Прототип страницы «Операции» в веб-программе Figma

Для проектирования дизайна была выбрана платформа Figma. Она предоставляет инструменты для прототипирования, которые позволяют

связать экраны и создать интерактивный прототип интерфейса. Figma хранит все проекты в облаке, что обеспечивает доступ к ним из любого места и с любого устройства. Она автоматически сохраняет изменения, что защищает от потери данных и гарантирует, что всегда есть актуальная версия проекта. Figma сохраняет историю версий проекта, позволяя вернуться к предыдущим состояниям, если это необходимо. Она позволяет легко экспортировать ресурсы (изображения, шрифты, CSS-стили) для разработчиков и предоставляет инструмент инспектирования, который позволяет разработчикам получать точную информацию о макете (размеры, цвета, шрифты и другое).

Вывод по разделу 2

Во втором разделе ВКР был осуществлен выбор методологии проектирования программного обеспечения, что является важным шагом для структурирования процесса разработки и обеспечения его соответствия требованиям проекта.

Для детального описания функциональных возможностей системы и разделения ролей пользователей веб-приложения разработана диаграмма вариантов использования. Проведено моделирование данных предметной области, что позволило определить ключевые сущности и связи между ними, необходимые для хранения и обработки информации.

Кроме того, спроектирован веб-интерфейс, учитывающий требования к удобству использования и визуальному представлению данных, что обеспечит интуитивно понятное взаимодействие пользователей с системой и повысит эффективность их работы. Были разработаны прототипы основных экранов, форм и элементов веб-интерфейса для лучшего пользовательского опыта. Использованы общепринятые шаблоны проектирования (для навигации, фильтрации, форм и других компонентов веб-приложения).

Все эти элементы в совокупности представляют собой детальную проектную документацию, необходимую для успешной реализации веб-приложения.

3 Реализация веб-приложения учёта финансов

3.1 Разработка диаграммы компонентов

Для наглядного представления взаимодействия различных частей приложения в рамках выбранной клиент-серверной архитектуры, предлагается построить диаграмму компонентов в нотации UML.

На рисунке 14 изображены четыре пакета, каждый из которых соответствует определённому слою. Слои взаимодействуют друг с другом. Внутри бизнес-слоя существуют компоненты, которые отвечают за определённые функции ПО, но компоненты не могут напрямую обратиться к базе данных. Взаимодействие осуществляется через слой доступа к данным.

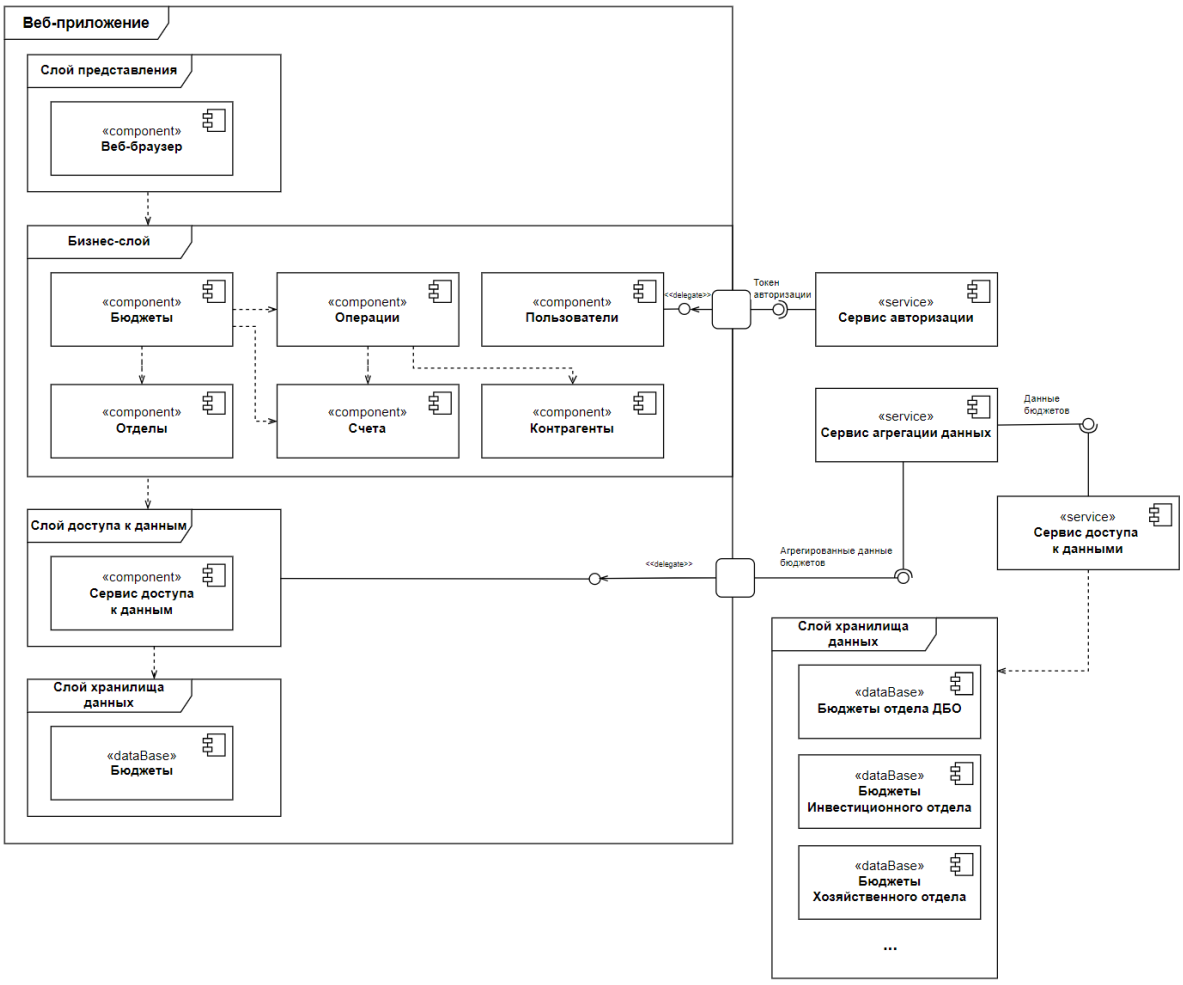


Рисунок 14 – Диаграмма компонентов разрабатываемого веб-приложения

Диаграмма компонентов – это визуальный инструмент, используемый для представления архитектуры программного обеспечения. Она наглядно показывает, из каких частей (компонентов) состоит программа и как эти части взаимодействуют между собой.

Данная диаграмма компонентов иллюстрирует многоуровневую архитектуру веб-приложения для работы с бюджетами, взаимодействующего с внешним сервисом авторизации и сервисом агрегации данных, который в свою очередь коммуницирует с сервисом доступа к данным организации (бюджеты разных департаментов). У веб-приложения есть внутренний слой хранения агрегированных данных с одной базой данной – бюджеты.

Внешние слои, с которыми взаимодействует веб-приложение:

- сервис авторизации: отдельный сервис, отвечающий за аутентификацию и авторизацию пользователей. Он предоставляет токены авторизации для доступа к ресурсам веб-приложения;
- сервис агрегации данных: отдельный сервис, отвечающий за сбор и обработку данных из разных источников слоя хранилища данных. Взаимодействует с сервисом доступа к данным;
- слой хранилища данных: централизованный сервер данных организации, состоящий из нескольких баз данных.

В ходе анализа и проектирования архитектуры веб-приложения, было принято решение о вынесении функционала по агрегации данных разных бюджетов в отдельный сервис, который будет отвечать за сбор, обработку и передачу данных о бюджетах веб-приложению.

Такое разделение на слои повышает устойчивость и позволяет легче внедрять изменения в отдельных частях системы, что в свою очередь улучшает масштабируемость, тестируемость, безопасность данных и обслуживание системы в целом.

3.2 Выбор средств реализации программного приложения

На следующем этапе выберем средства реализации программного приложения. Данный этап является очень важным, так как сделав правильный выбор, можно добиться не только быстрой и удобной реализации программного обеспечения, но и повысить его работоспособность. Согласно рейтингам Tiobe и PYPL (индексы, оценивающие популярность языков программирования, на основе подсчёта результатов поисковых запросов, содержащих название языка) Python, Java, JavaScript и C# являются самыми популярными языками программирования (данные на май 2024 года) [3]. Сравним их в таблице 4 для выбора наиболее подходящего языка для разработки веб-приложения учета финансов.

Таблица 4 – Сравнительный анализ популярных языков программирования для серверных приложений

Параметр сравнения	Java	C#	Python	Javascript
Надежность и безопасность	Высокая. Статическая типизация, надежная JVM, проверенный временем.	Высокая. Статическая типизация, надежная среда выполнения.	Средняя, но существуют методы повышения безопасности. Динамическая типизация.	Низкая. Динамическая типизация, больше рисков runtime ошибок.
Производительность	Высокая, благодаря JVM.	Высокая. Компилируемый язык (или с компиляцией в IL).	Средняя. Интерпретируемый язык, может быть медленнее Java и C#.	Низкая. Может быть достаточно быстрым только в браузере.
Масштабируемость	Высокая. Поддержка многопоточности, распределенных систем и микросервисов.	Высокая. .NET framework/core предоставляет инструменты для построения масштабируемых приложений.	Средняя. Имеет решения для масштабирования, но не всегда так эффективно, как Java и C#.	Средняя. В основном используется для front-end, бэкенд-реализации могут быть не всегда масштабируемы.

Продолжение таблицы 4

Параметр сравнения	Java	C#	Python	Javascript
Универсальность	Высокая. Применяется не только для бэкенда, но и для мобильных, desktop-приложений.	Высокая. Применяется не только для бэкенда, но и для мобильных, desktop-приложений.	Высокая. Широко используется для бэкенда, data science, scripting.	Высокая. Чаще для front-end, но все чаще и для бэкенда (Node.js).
Время разработки	Среднее. Возможно, больше, чем в Python, из-за большей сложности структуры.	Среднее. Понятный синтаксис, но в некоторых случаях может быть больше кода.	Низкое. Простой синтаксис, много готовых библиотек.	Низкое. Быстро осваивается, много готовых фреймворков.
Сложность обучения	Средняя. Необходимо понимание концепций ООП.	Средняя. Необходимо понимание концепций ООП.	Низкая. Легко осваивается, чистый синтаксис.	Низкая. Быстро осваивается, простая структура.
Поддержка в банковском секторе	Высокая. Многолетняя история использования в финансовых системах.	Высокая. .NET имеет активное использование и поддержку в финансовом секторе.	Низкая. Не так часто используется в критически важных банковских системах.	Низкая. Не так часто используется в критически важных банковских системах.

В контексте бэкенда веб-приложения учета финансов, Java по-прежнему обладает хорошим балансом между производительностью, надежностью, удобством разработки и поддержки со стороны сообщества. C# может быть слишком сложным и не всегда необходимым для этой задачи, хотя и обладает максимальной производительностью.

Рассмотрим другие технологии, необходимые для разработки веб-приложения на языке Java.

Spring Boot – упрощает создание приложений на Java, благодаря автоконфигурации, встроенному серверу и готовым стартерам. Это

значительно ускорит процесс разработки, позволит команде сосредоточиться на бизнес-логике.

Spring Data JPA упростит работу с реляционными базами данных, избавит от написания большого количества шаблонного кода. Spring Data JPA обеспечивает ORM, что позволит работать с данными как с объектами Java, не прибегая к прямому SQL.

Spring Web обеспечит простую реализацию RESTful API, что позволит создать бэкенд, который легко интегрируется с фронтендом. Он включает встроенный веб-сервер (Tomcat, Jetty, Undertow или Netty), что упрощает развертывание приложения [20].

Spring Validation позволяет легко валидировать входные данные с помощью аннотаций (например, @NotNull, @Size, @Email). Гарантирует, что данные, которые сохраняются в базе данных, соответствуют заданным ограничениям.

Lombok автоматически генерирует getter’ы, setter’ы, конструкторы и другие методы, что значительно сокращает объем boilerplate-кода.

Apache POI обеспечивает возможность создавать и обрабатывать файлы Excel (.xlsx), библиотека полезна для экспорта отчетов или загрузки данных.

JUnit является стандартом для написания модульных тестов в Java. JUnit помогает гарантировать качество кода, выявляя ошибки на ранних этапах разработки. JUnit позволяет уверенно проводить рефакторинг, гарантируя, что изменения не сломают существующий функционал.

Сравним популярные фронтенд-фреймворки в таблице 5 для выбора наиболее подходящего для разработки веб-приложения учета финансов.

Таблица 5 – Сравнительный анализ популярных фреймворков для разработки веб-интерфейсов [19]

Параметр сравнения	Angular	React	Vue
Архитектура	Компонентная, с сильным управлением зависимостями и «двусторонним связыванием». Встроенная инъекция зависимостей.	Компонентная, «одностороннее связывание»	Компонентная, «одностороннее связывание»
Управление данными	Использует компоненты, сервисы, RxJS для управления данными. Сильное двустороннее связывание данных делает взаимодействие с данными более прямым.	Использует состояния (state), управление данными основано на принципах «одностороннего связывания». Обычно применяется Redux, MobX или другие библиотеки для более сложных приложений.	Использует состояния (state), управление данными основано на принципах «одностороннего связывания». Есть библиотеки, аналогичные Redux, но более лаконичные решения доступны.
Производительность	Хорошая производительность, особенно для крупных приложений, обеспечивается правильным использованием принципов Angular.	Производительность на среднем уровне, и за ней нужно хотя бы изредка следить – как минимум встречаются лишние ререндеринги компонентов.	Хорошая производительность благодаря встроенной системе реактивности и компилятору шаблонов.
Масштабируемость	Высокая, обеспечивается модульной архитектурой и четкой организацией кода.	Высокая, хорошо подходит для крупных приложений.	Высокая для средних проектов, но для очень крупных может потребоваться больше ресурсов.
Обучение	Относительно высокое, требуется изучение TypeScript и Angular-специфичных концепций.	Среднее, требуется изучение TypeScript и React-специфичных концепций.	Среднее, требуется изучение TypeScript и Vue-специфичных концепций.
Сообщество и поддержка	Высокое. Активное сообщество, обширная документация и поддержка.	Высокое. Активное, обширное сообщество, большая поддержка.	Меньшая экосистема по сравнению с Angular или React.

Для приложения учета финансов, требующего высокой надежности, выбор Angular может оказаться предпочтительным, несмотря на более высокую кривую обучения, чем React или Vue. Строгая типизация в TypeScript помогает предотвратить ошибки на ранних стадиях разработки, что критично для финансовых приложений. Модульная архитектура, «двустороннее связывание» и инструменты управления состоянием (RxJS), хорошо подходят для сложных бизнес-логик, типичных для финансовых приложений. Благодаря наличию готовых UI-компонентов, Angular и PrimeNG дают возможность быстрее вывести на рынок качественный продукт.

RxJS обеспечивает реактивный подход к работе с асинхронными данными и потоками событий, что упрощает работу с сетевыми запросами и другими асинхронными операциями. RxJS предоставляет широкий набор операторов для преобразования, фильтрации и управления потоками данных.

PrimeNG предоставляет широкий набор готовых UI-компонентов (таблицы, формы, кнопки, модальные окна и другие), что значительно ускоряет разработку интерфейса. PrimeNG предлагает настраиваемый дизайн, что позволяет адаптировать компоненты под общий стиль приложения.

Chart.js позволяет легко создавать графики и диаграммы для визуализации финансовых данных. Chart.js поддерживает множество типов графиков (линейные, столбчатые, круговые и другие). Chart.js прост в использовании и имеет понятный API.

Karma – это тестовый runner, который запускает тесты, а Jasmine – фреймворк для написания тестов в Angular. Karma и Jasmine позволяют писать модульные тесты для Angular-компонентов, что повышает качество кода и обеспечивает уверенность при рефакторинге.

IntelliJ IDEA для бэкенда и WebStorm для фронтенда – обеспечивают наилучшие возможности для разработки веб-приложения учета финансов.

Обе IDE предоставляют инструменты, которые позволяют повысить производительность, улучшить качество кода и сократить время разработки.

На рисунке 15 представлена программная архитектура разрабатываемого веб-приложения.

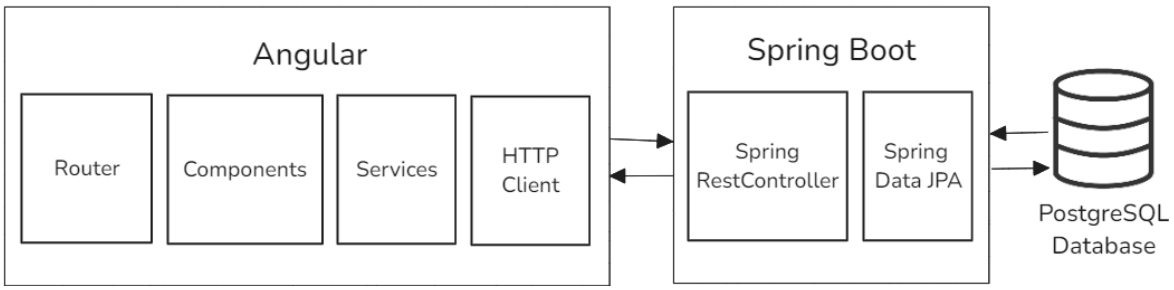


Рисунок 15 – Программная архитектура разрабатываемого веб-приложения

Все выбранные технологии отлично сочетаются друг с другом, предоставляя мощный и надежный инструментарий для создания современного веб-приложения учета финансов. Они позволяют создавать как функциональный и надежный бэкенд, так и интерактивный и удобный фронтенд. Этот стек технологий также обеспечивает высокую производительность, масштабируемость и гибкость, что особенно важно для такого сложного и критически важного приложения, как система учета финансов.

3.3 Создание базы данных

IntelliJ IDEA предоставляет все необходимые инструменты для работы с базами данных, включая PostgreSQL, прямо из среды разработки. Это ускоряет процесс разработки, так как нам не нужно переключаться между различными приложениями для работы с кодом и базами данных.

Рассмотрим созданные таблицы в базе данных «budgets» для веб-приложения учета финансов на рисунке 16.

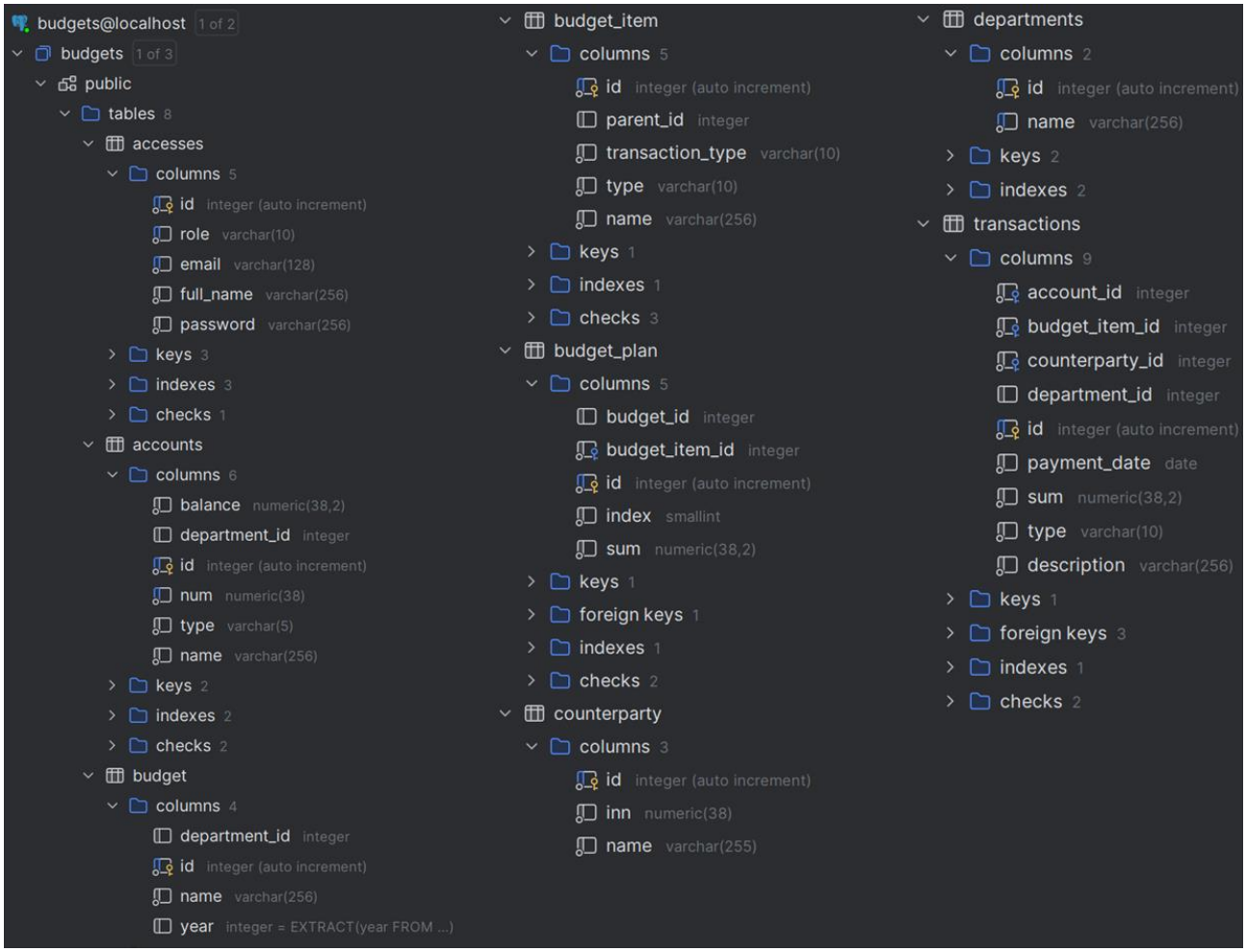


Рисунок 16 – Таблицы базы данных «budgets» разрабатываемого веб-приложения в IDEA

IDEA позволяет просматривать таблицы, схемы, индексы и другие объекты базы данных. Также она может синхронизировать структуру базы данных с кодом на языке Java, что упрощает работу с ORM-фреймворками.

3.4 Алгоритмы обработки данных

Все основные вычисления и бизнес-логика приложения будут производится на серверной стороне. Рассмотрим самые сложные алгоритмы в программе в виде блок-схем. Блок-схема алгоритма – это визуальное представление, состоящее из связанных графических символов, каждый из

которых отображает определенный этап обработки данных, а линии потока указывают на порядок их выполнения. Эти символы, называемые операционными блоками, наглядно иллюстрируют логику и последовательность шагов, необходимых для решения задачи или выполнения процесса [4]. На рисунке 17 представим алгоритм создания бюджета.

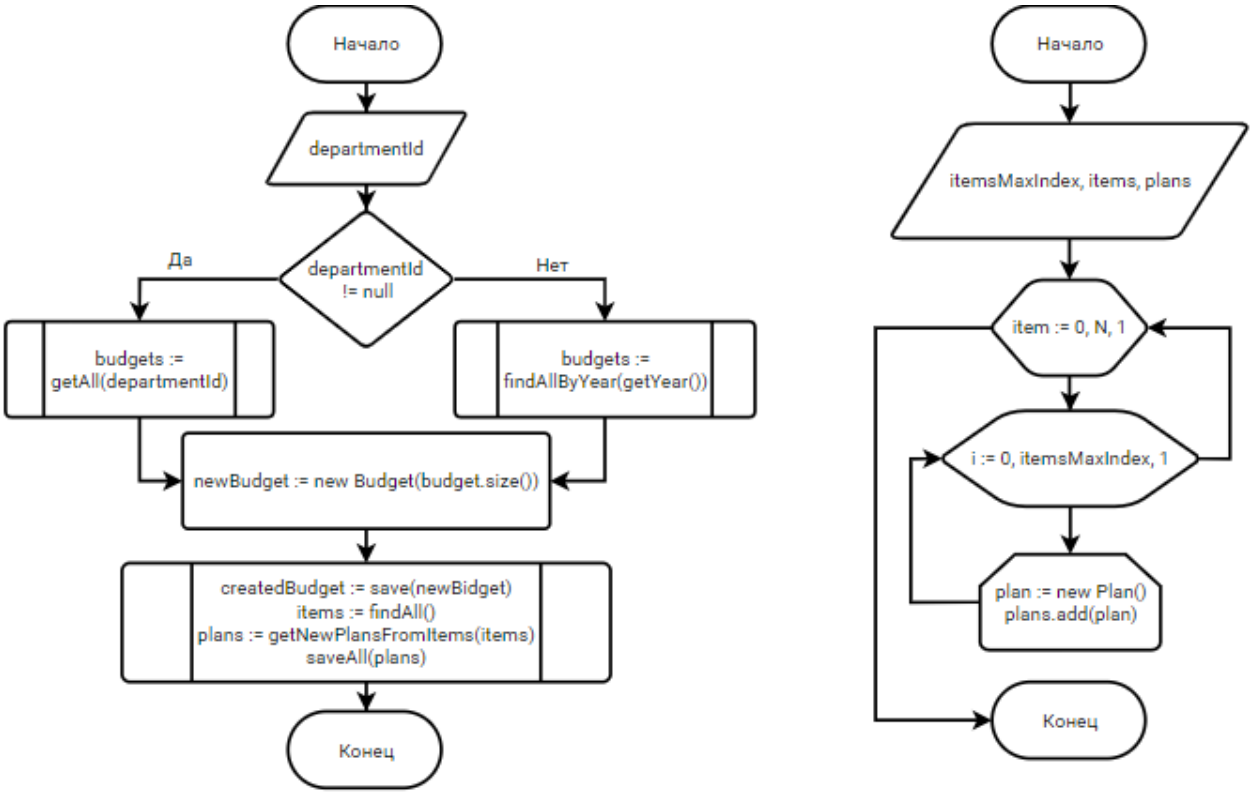


Рисунок 17 – Алгоритм создания бюджета

На вход программе передаётся аргумент departmentId, если он равен null тогда вызывается функция findAllByYear, которой на вход передаётся текущий год. Если же departmentId не равен null, тогда вызывается функция findAll, которая на вход принимает аргумент departmentId. Обе функции на выходе возвращают найденные бюджеты.

Затем создаётся новый бюджет, запрашиваются все статьи бюджета и вызывается функция getNewPlansFromItems, в которую передаются ID

созданного бюджета и статьи бюджета, а на выходе она возвращает планы бюджета, заполненные нулями, которые сохраняются в БД budget_plan.

Справа, на рисунке 17 представлен алгоритм работы метода getNewPlansFromItems. На входе программы имеется три переменные, константа itemsMaxIndex, которая равна 13 (индекс месяцев начинается с 0, а 12 индекс нужен для сохранения итоговой суммы), переменная items – полученная из БД со статьями бюджета и plans, которая инициализируется пустым массивом, для дальнейшего его заполнения. Далее программа входит в цикл по элементам массива items с вложенным циклом, который бежит 13 раз до константы itemsMaxIndex и на каждой итерации создаётся новый план и добавляется в массив plans.

В коде, написанном на языке Java с использованием фреймворка Spring Boot описанные алгоритмы реализованы следующим образом, как представлено на рисунках 18 и 19.

```
@Transactional  Alexander Chistov *
@PostMapping(consumes = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<Integer> create(@RequestBody Map<String, Integer> body) {
    Integer departmentId = body.get("departmentId");
    log.info("create with departmentId = {}", departmentId);

    List<Budget> budgets = (departmentId != null) ?
            repository.getAllByDepartmentId(departmentId) :
            repository.findAllByYear(LocalDate.now().getYear());

    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd.MM.yyyy");
    Budget newBudget = new Budget(
            id: null,
            name: "Версия №" + (budgets.size() + 1) + " от " + LocalDate.now().format(formatter),
            departmentId,
            LocalDate.now().getYear()
    );

    Budget createdBudget = repository.save(newBudget);

    List<BudgetItem> items = itemRepository.findAll();

    List<BudgetMonth> plans = getNewPlansFromItems(createdBudget.getId(), items);

    planRepository.saveAll(plans);

    return new ResponseEntity<>(createdBudget.getId(), HttpStatus.CREATED);
}
```

Рисунок 18 – Метод создания нового бюджета на языке Java

```java
public static List<BudgetMonth> getNewPlansFromItems(int budgetId, List<BudgetItem> items) {
    int BUDGETS_ITEMS_MAX_INDEX = 13;
    List<BudgetMonth> plans = new ArrayList<>();

    for (BudgetItem item : items) {
        for (byte i = 0; i < BUDGETS_ITEMS_MAX_INDEX; i++) {
            BudgetMonth plan = new BudgetMonth( id: null, i, BigDecimal.ZERO, budgetId, item);
            plans.add(plan);
        }
    }

    return plans;
}
```

Рисунок 19 – Метод заполнения массива plans объектами класса BudgetMonth

На рисунке 20 рассмотрим алгоритм обновления данных уже существующего бюджета.

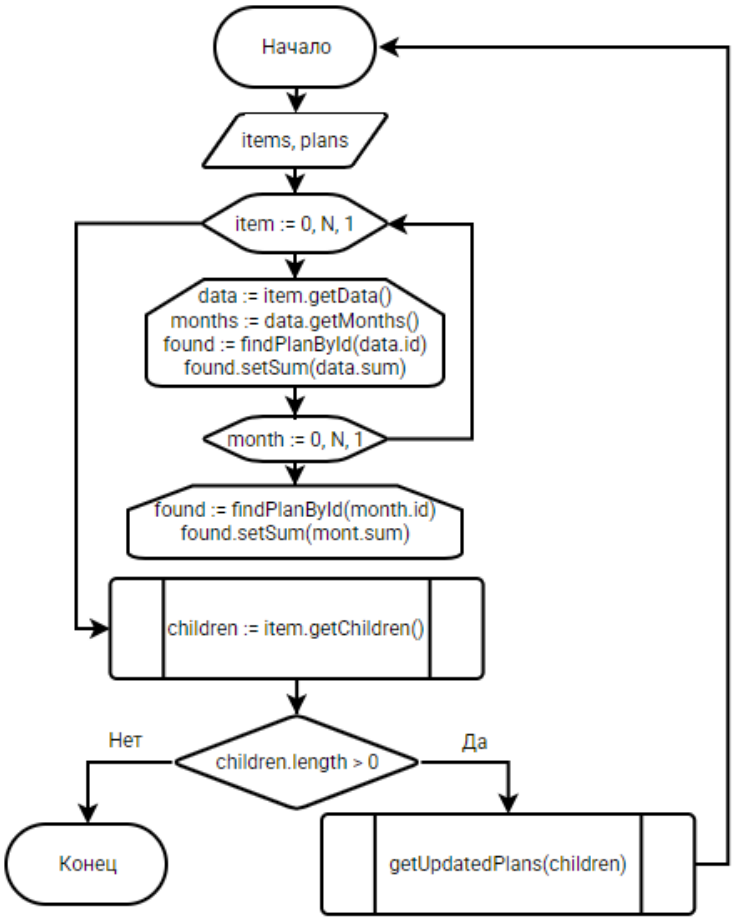


Рисунок 20 – Алгоритм обновления данных бюджета

На вход программа принимает два аргумента – массивы items и plans. Затем программа входит в цикл по массиву элементов items. В теле цикла программа находит объект план по ID текущего элемента и устанавливает в него значение – сумму текущего элемента. Затем запускается вложенный цикл по массиву элементов months. В теле цикла программа находит план по ID текущего элемента и устанавливает в него значение – сумму текущего элемента. Далее из текущего элемента запрашиваются дочерние элементы, если есть хотя бы один дочерний элемент – программа рекурсивно запускает саму себя с передачей массива дочерних элементов.

В коде, написанном на языке Java с использованием фреймворка Spring Boot данный алгоритм реализован следующим образом, как представлено на рисунках 21 и 22.

```
@Transactional  Alexander Chistov
@PutMapping(value = "/{budgetId}", consumes = MediaType.APPLICATION_JSON_VALUE)
@ResponseStatus(HttpStatus.NO_CONTENT)
public void update(@Valid @RequestBody BudgetDto budgetDto, @PathVariable int budgetId) {
    log.info("update {} with budgetId={}", budgetDto, budgetId);
    assureIdConsistent(budgetDto, budgetId);

    List<BudgetItemDto> items = budgetDto.getBudgetItems();
    List<BudgetMonth> plans = planRepository.getAllByBudgetId(budgetId);

    List<BudgetMonth> updatedPlans = getUpdatedBudgetMonths(items, plans);

    planRepository.saveAll(updatedPlans);
}
```

Рисунок 21 – Метод обновления данных бюджета в Java контроллере

```java
public static List<BudgetMonth> getUpdatedBudgetMonths(List<BudgetItemDto> items, List<BudgetMonth> plans) {
    for (BudgetItemDto item : items) {
        BudgetDataDto data = item.getData();
        BudgetDataMonthDto[] dataMonths = data.getMonths();

        Arrays.stream(dataMonths).forEach( BudgetDataMonthDto month -> {
            BudgetMonth found = findPlanById(plans, month.getId());
            found.setSum(month.getPlan());
        });

        BudgetMonth found = findPlanById(plans, data.getId());
        found.setSum(data.getPlanTotal());

        List<BudgetItemDto> childrenItems = item.getChildren();

        if (childrenItems != null && !childrenItems.isEmpty()) {
            getUpdatedBudgetMonths(childrenItems, plans);
        }
    }

    return plans;
}
```

Рисунок 22 – Метод обновления данных бюджета в утилитарном классе

Далее рассмотрен контрольный пример реализации проекта.

3.5 Контрольный пример реализации проекта

При первом открытии веб-приложения появится форма входа (рисунок 23). Необходимо ввести email и пароль от входа в систему выданные администратором и нажать кнопку «Войти».

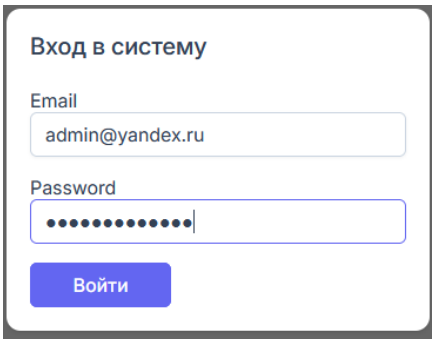


Рисунок 23 – Вход в веб-приложение учета финансов

После успешного входа в систему пользователь будет перенаправлен системой на страницу «Отчеты». В процессе загрузки страницы будет отображаться заполнитель, как показано на рисунке 24. После получения всех данных, на странице отобразятся результаты.

Рассмотрим пользовательский интерфейс веб-приложения учёта финансов. Навигация по приложению осуществляется через главное меню, обеспечивающее быстрый доступ к основным разделам («Отчеты», «Операции», «Бюджеты», «Управление доступом»). Раздел «Отчеты» предоставляет пользователям возможность просматривать отчеты по доходам или расходам (рисунок 25) учитывая выбранную версию бюджета. Отчеты содержат ключевые показатели эффективности бизнеса, представленные в разрезе месяцев и в виде итоговых сводных данных.

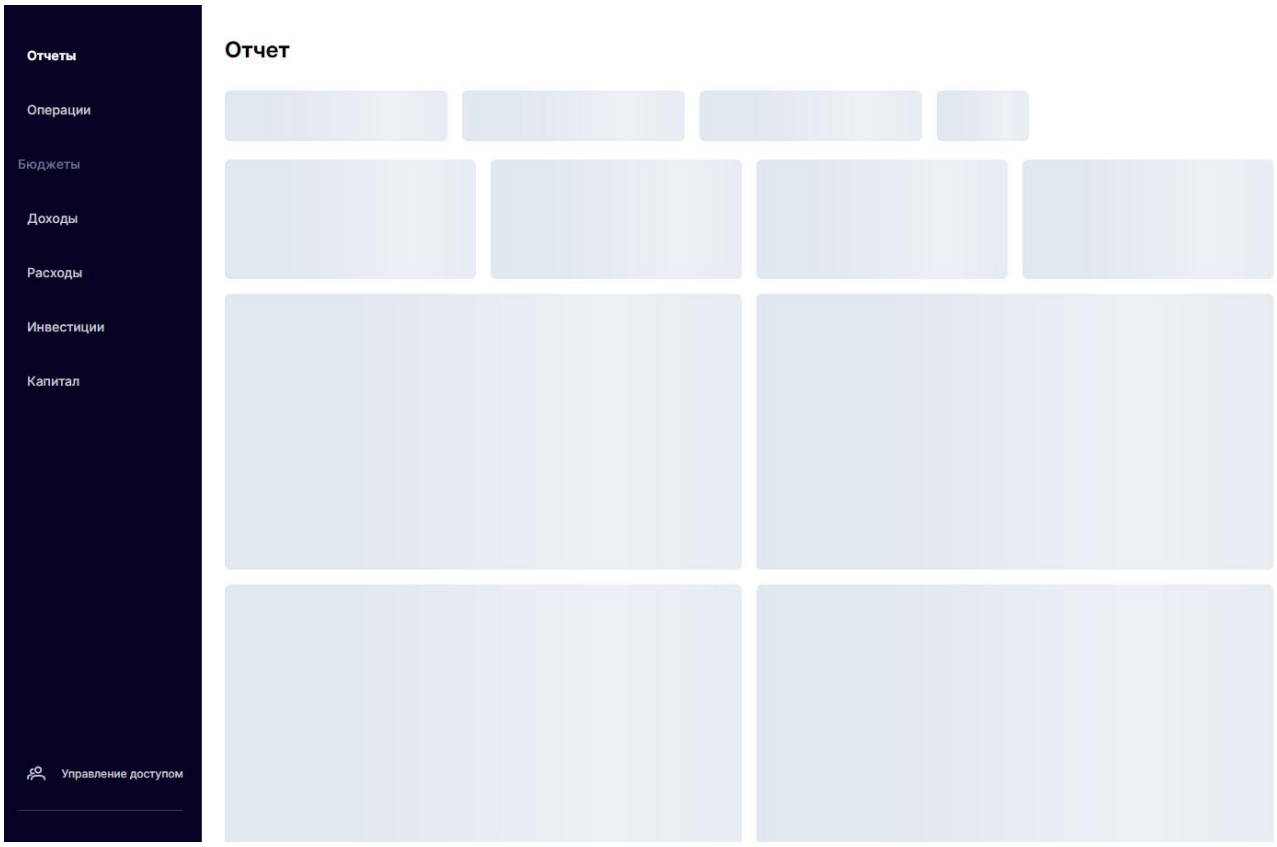


Рисунок 24 – Загрузка страницы «Отчеты»

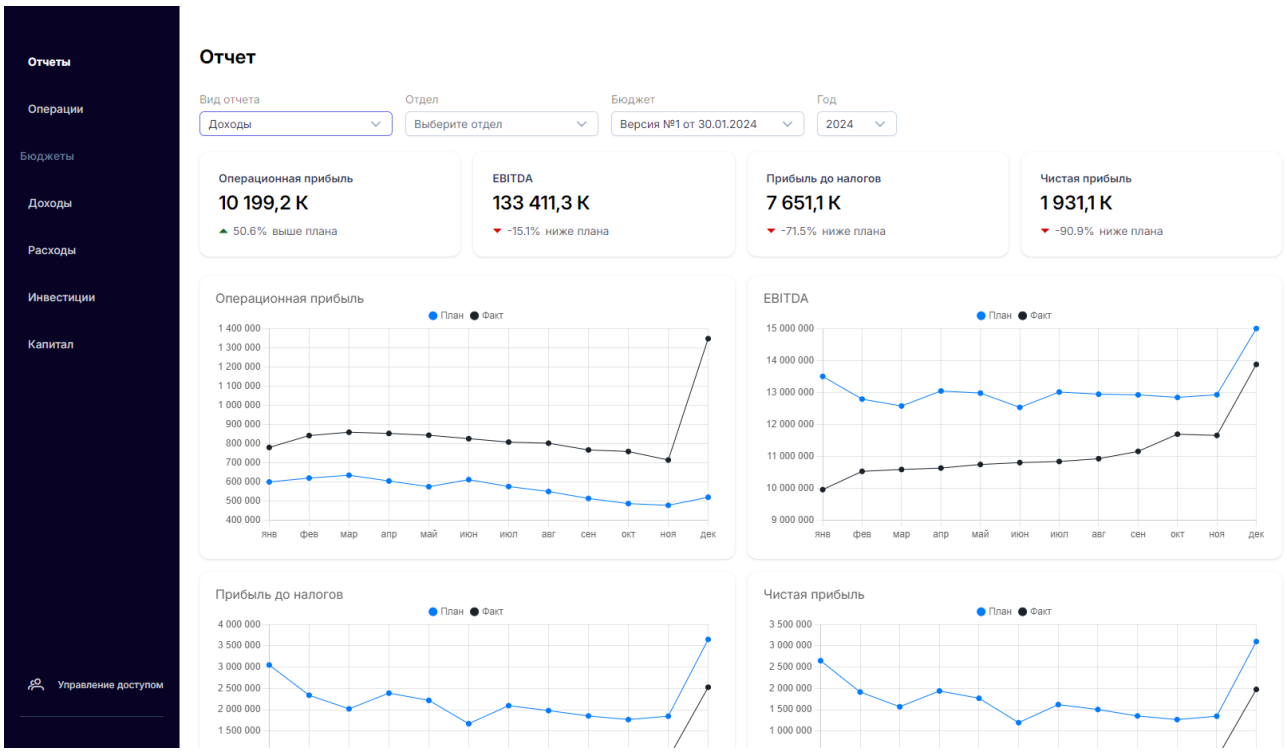


Рисунок 25 – Раздел «Отчеты» по доходам

Раздел «Операции» (рисунок 26) предназначен для управления денежными операциями в организации. Здесь пользователь может не только просматривать историю всех совершенных транзакций, но и добавлять новые, а также удалять существующие операции.

Раздел «Бюджеты» позволяет создавать новые бюджеты, редактировать существующие и удалять неактуальные. Приложение поддерживает ручной ввод плановых сумм, а также импорт и экспорт бюджетов в формате Excel, обеспечивая удобство и гибкость в работе с финансовыми данными. На рисунке 27 представлен подраздел «Капитал» раздела «Бюджеты».

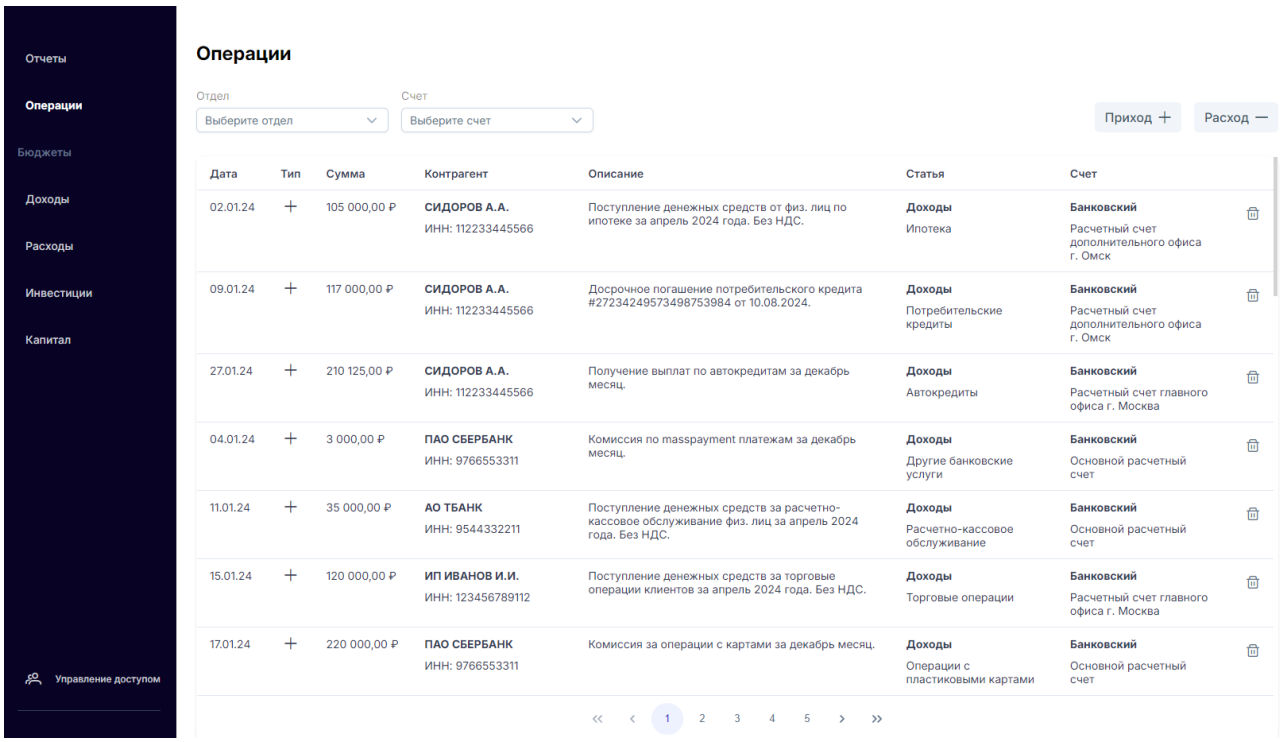


Рисунок 26 – Раздел «Операции»

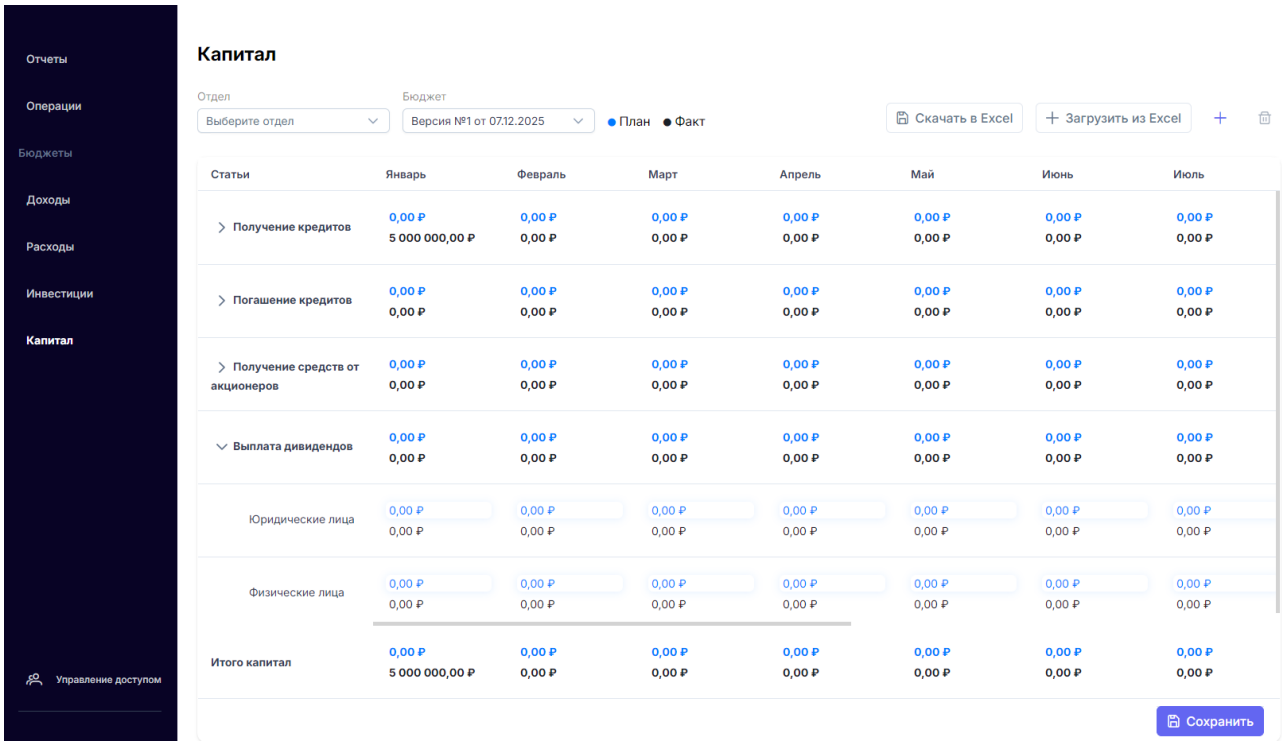


Рисунок 27 – Раздел «Бюджеты» по капиталу

Рассмотрим типовые задачи, с которыми ежедневно сталкиваются пользователи веб-приложения. Для того, чтобы добавить новую операцию (например, расход, доход или инвестицию) на странице «Операции» необходимо нажать на одну из кнопок «Приход» или «Расход», в зависимости от типа операции, которую пользователь желает добавить, как показано на рисунке 28.

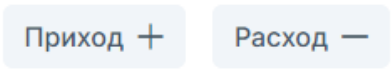


Рисунок 28 – Кнопки добавления денежной операции в разделе «Операции»

Затем нужно заполнить открывшуюся форму данными и нажать кнопку «Добавить», как показано на рисунке 29.

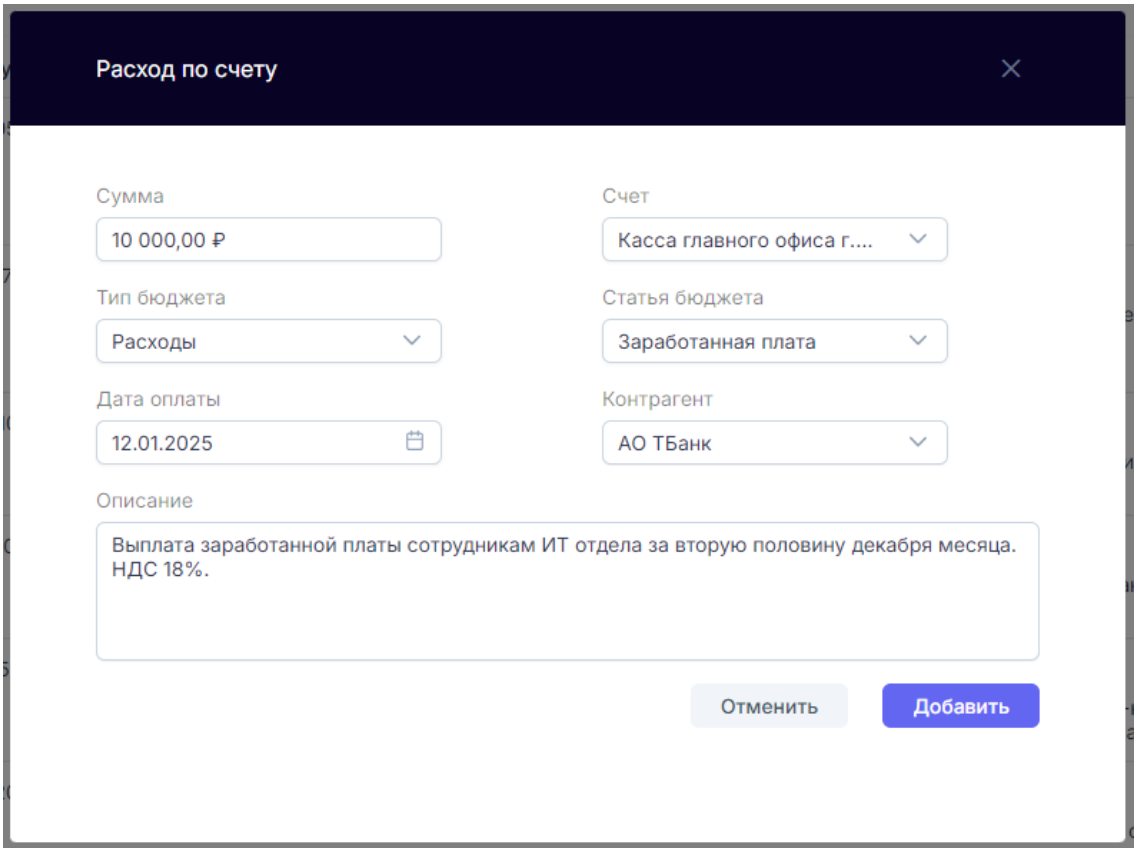


Рисунок 29 – Форма добавления денежной операции в разделе «Операции»

Добавленная операция отобразится на странице в списке операций последней. Для того, чтобы отфильтровать операции в списке, нужно выбрать отдел и счет, привязанный к выбранному отделу (рисунок 30). После этого на странице отобразятся соответствующие фильтрам операции.

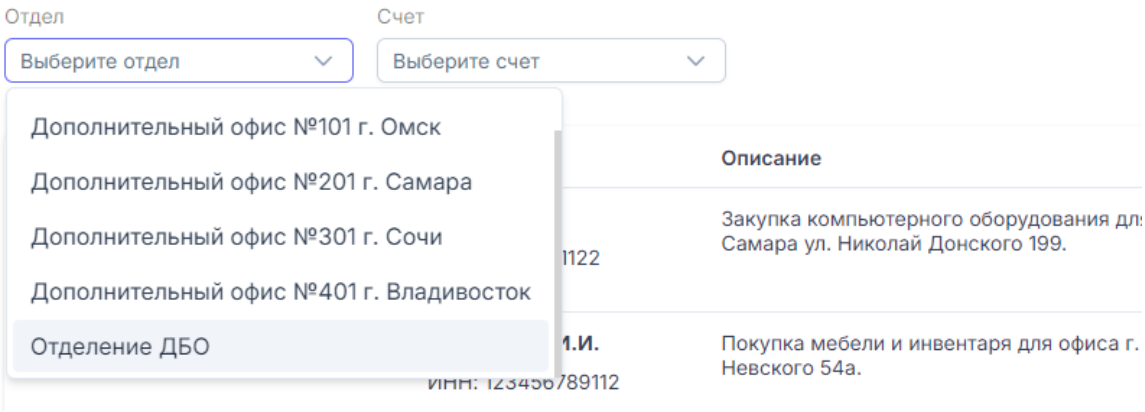


Рисунок 30 – Фильтрация денежных операций в разделе «Операции»

Для создания нового бюджета, необходимо нажать иконку «+» в правом верхнем углу на странице «Бюджеты», как показано на рисунке 31.

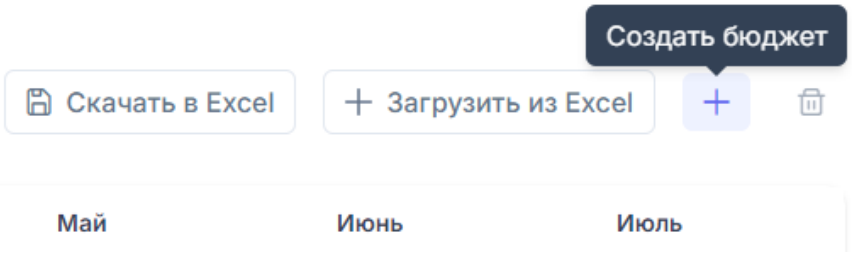


Рисунок 31 – Создание нового бюджета в разделе «Доходы»

Для того, чтобы отредактировать уже существующий бюджет, необходимо внести нужную сумму в соответствующую ячейку и нажать кнопку «Сохранить», как показано на рисунке 32.

Рисунок 32 – Редактирование существующего бюджета в разделе «Доходы»

Для удаления уже существующего бюджета, необходимо нажать иконку «Корзина» в правом верхнем углу на странице «Бюджеты», как показано на рисунке 33. Для того, чтобы скачать существующий бюджет нужно нажать кнопку «Скачать в Excel.

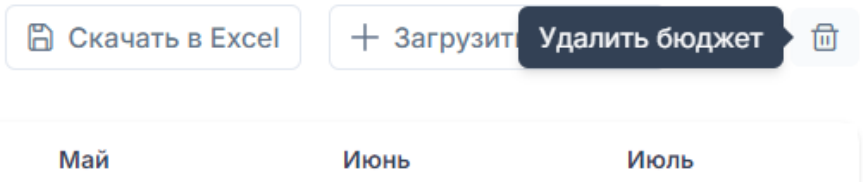
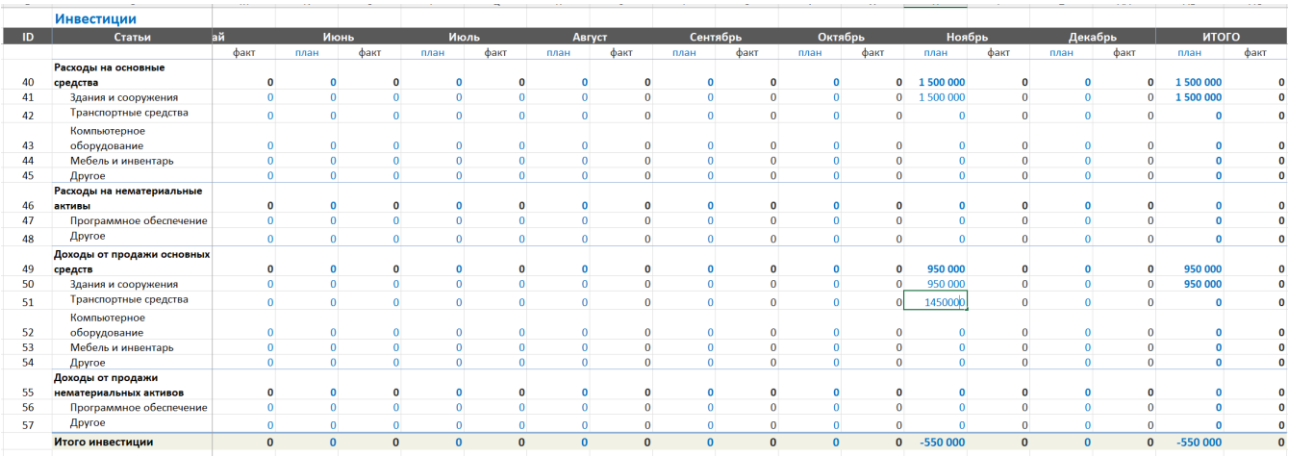


Рисунок 33 – Удаление существующего бюджета в разделе «Доходы»

После этого файл скачается в указанную папку, как правило эта папка называется «Загрузки».

Для того, чтобы изменить текущий бюджет используя файл Excel, необходимо нажать кнопку «Скачать в Excel», заполнить данные скачанного шаблона, как показано на рисунке 34.



Инвестиции

ID	Статьи	ай		Июнь		Июль		Август		Сентябрь		Октябрь		Ноябрь		Декабрь		ИТОГО	
		факт	план	факт	план	факт	план	факт	план	факт	план	факт	план	факт	план	факт	план	факт	
40	**Расходы на основные средства**	0	0	0	0	0	0	0	0	0	0	0	0	1 500 000	0	0	0	1 500 000	0
41	Здания и сооружения	0	0	0	0	0	0	0	0	0	0	0	0	1 500 000	0	0	0	1 500 000	0
42	Транспортные средства	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
43	Компьютерное оборудование	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
44	Мебель и инвентарь	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
45	Другое	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
46	**Расходы на нематериальные активы**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
47	Программное обеспечение	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
48	Другое	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
49	**Доходы от продажи основных средств**	0	0	0	0	0	0	0	0	0	0	0	0	950 000	0	0	0	950 000	0
50	Здания и сооружения	0	0	0	0	0	0	0	0	0	0	0	0	950 000	0	0	0	950 000	0
51	Транспортные средства	0	0	0	0	0	0	0	0	0	0	0	0	1450000	0	0	0	0	0
52	Компьютерное оборудование	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
53	Мебель и инвентарь	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
54	Другое	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
55	**Доходы от продажи нематериальных активов**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
56	Программное обеспечение	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
57	Другое	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Итого инвестиции	0	0	0	0	0	0	0	0	0	0	0	0	-550 000	0	0	0	-550 000	0

Рисунок 34 – Файла-шаблон Excel для типа бюджета «Инвестиции»

Затем загрузить этот файл нажав кнопку «Загрузить из Excel». Данные автоматически рассчитаются, сохранятся в базу данных и отобразят в веб-интерфейсе пользователя.

Если в процессе работы с веб-интерфейсом сервер успешно обработает данные или вернет ошибку, пользователь получит об этом уведомление, в правом верхнем углу страницы, как показано на рисунке 35.

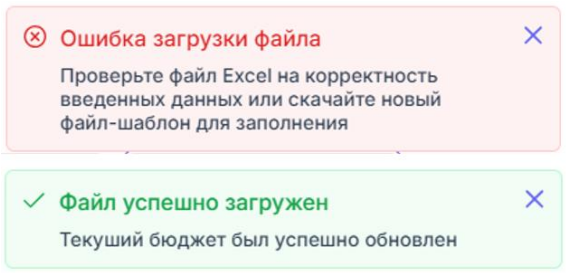


Рисунок 35 – Уведомления о загрузке файла Excel для бюджета «Доходы»

Добавить нового пользователя может только пользователь с правами администратора. Для этого, в разделе «Управление доступом», необходимо заполнить обязательные поля и нажать кнопку «Добавить» (рисунок 36).

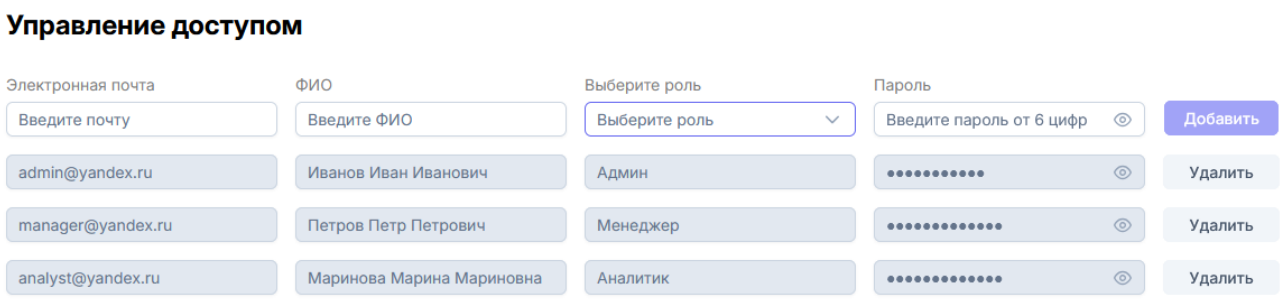


Рисунок 36 – Раздел «Управление доступом»

3.6 Тестирование базы данных и прикладных программ

На рисунке 37 представлены тестовые данные, которые необходимы для проверки работоспособности основных функций разрабатываемой информационной системы.

```
INSERT INTO accesses (full_name, email, password, role)
VALUES ( full_name 'Иванов Иван Иванович', email 'admin@yandex.ru', password '{noop}admin', role 'ADMIN'),
       ( full_name 'Петров Петр Петрович', email 'manager@yandex.ru', password '{noop}manager', role 'MANAGER'),
       ( full_name 'Маринова Марина Мариновна', email 'analyst@yandex.ru', password '{noop}analyst', role 'ANALYST');

INSERT INTO departments (name)
VALUES ( name 'Главный офис №1 г. Москва'),
       ( name 'Дополнительный офис №101 г. Омск'),
       ( name 'Дополнительный офис №201 г. Самара'),
       ( name 'Дополнительный офис №301 г. Сочи'),
       ( name 'Дополнительный офис №401 г. Владивосток'),
       ( name 'Отделение ДБО');

INSERT INTO counterparty (name, inn)
VALUES ( name 'ИП Иванов И.И.', inn 123456789112),
       ( name 'Федеральная налоговая служба', inn 9876543211),
       ( name 'Центральный банк России', inn 9988776655),
       ( name 'ПАО Сбербанк', inn 9766553311),
       ( name 'ООО Яндекс', inn 9655442211),
       ( name 'АО ТБанк', inn 9544332211),
       ( name 'Центральное ЖКХ г. Москва', inn 9122334455),
       ( name 'Сидоров А.А.', inn 112233445566),
       ( name 'ПАО МТС', inn 9433221122),
       ( name 'ПАО Ростелеком', inn 9322112233);

INSERT INTO accounts (num, type, balance, name, department_id)
VALUES ( num 1111111111111111, type 'BANK', balance 95000775.85, name 'Основной расчетный счет', department_id null),
       ( num 1234123412341234, type 'BANK', balance 10550990.65, name 'Расчетный счет главного офиса г. Москва', department_id 1),
       ( num 1212121212121212, type 'CASH', balance 770150.05, name 'Касса главного офиса г. Москва', department_id 1),
       ( num 3323332233223322, type 'BANK', balance 1050950, name 'Расчетный счет дополнительного офиса г. Омск', department_id 2),
       ( num 3333333233323332, type 'CASH', balance 365695.95, name 'Касса дополнительного офиса г. Омск', department_id 2),
       ( num 1122112211221122, type 'BANK', balance 2196475.15, name 'Расчетный счет ДБО', department_id 6);

INSERT INTO budget_item (type, transaction_type, name, parent_id)
VALUES ( type 'REVENUE', transaction_type 'INCOME', name 'Доходы от физических лиц', parent_id null),
       ( type 'REVENUE', transaction_type 'INCOME', name 'Кредитование', parent_id 1),
       ( type 'REVENUE', transaction_type 'INCOME', name 'Ипотека', parent_id 2),
       ( type 'REVENUE', transaction_type 'INCOME', name 'Потребительские кредиты', parent_id 2),
       ( type 'REVENUE', transaction_type 'INCOME', name 'Автокредиты', parent_id 2),
```

Рисунок 37 – Тестовые данные для загрузки в базы данных в IDEA

В Spring Boot приложении файл data.sql используется для инициализации базы данных при запуске приложения. Этот файл содержит SQL-запросы, которые выполняются автоматически после того, как Spring Boot успешно подключится к базе данных [18].

Модульное, или юнит тестирование позволяет тестировать отдельные модули (компоненты, классы, методы) приложения изолированно друг от друга. Это помогает выявлять ошибки на ранних этапах разработки и не допускать распространения ошибок по всему приложению. Юнит-тесты дешевле и быстрее других. Когда тест падает, это указывает на конкретный модуль, что упрощает поиск и исправление ошибок [15].

Модульные тесты можно автоматизировать, что позволяет запускать их при каждой сборке или изменении кода, обеспечивая постоянный контроль качества. Модульные тесты могут служить в качестве документации, описывая как должен работать тот или иной модуль.

Прежде чем начать тестирование, подготовим тестовые данные, представленные на рисунке 38, максимально приближенные к реальным условиям использования приложения, чтобы обеспечить наиболее точную оценку его работы.

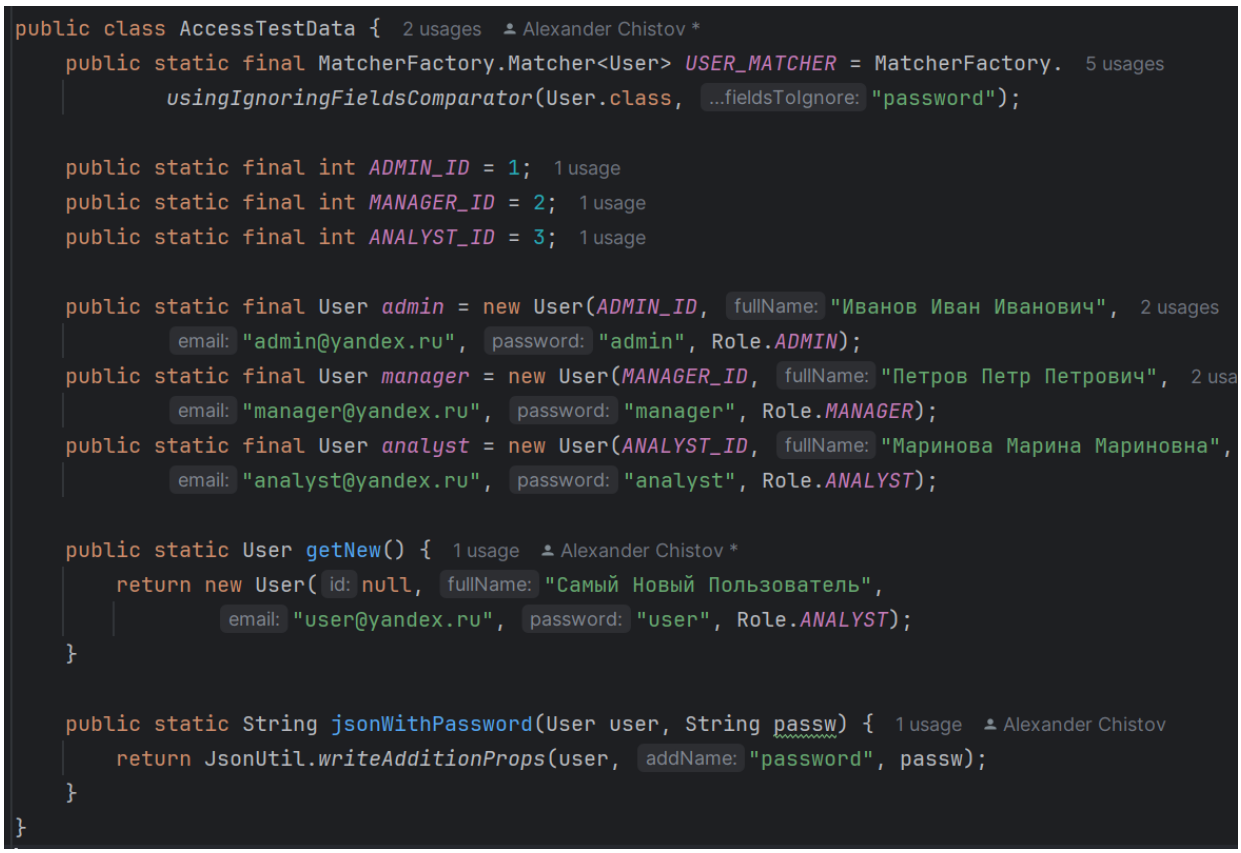
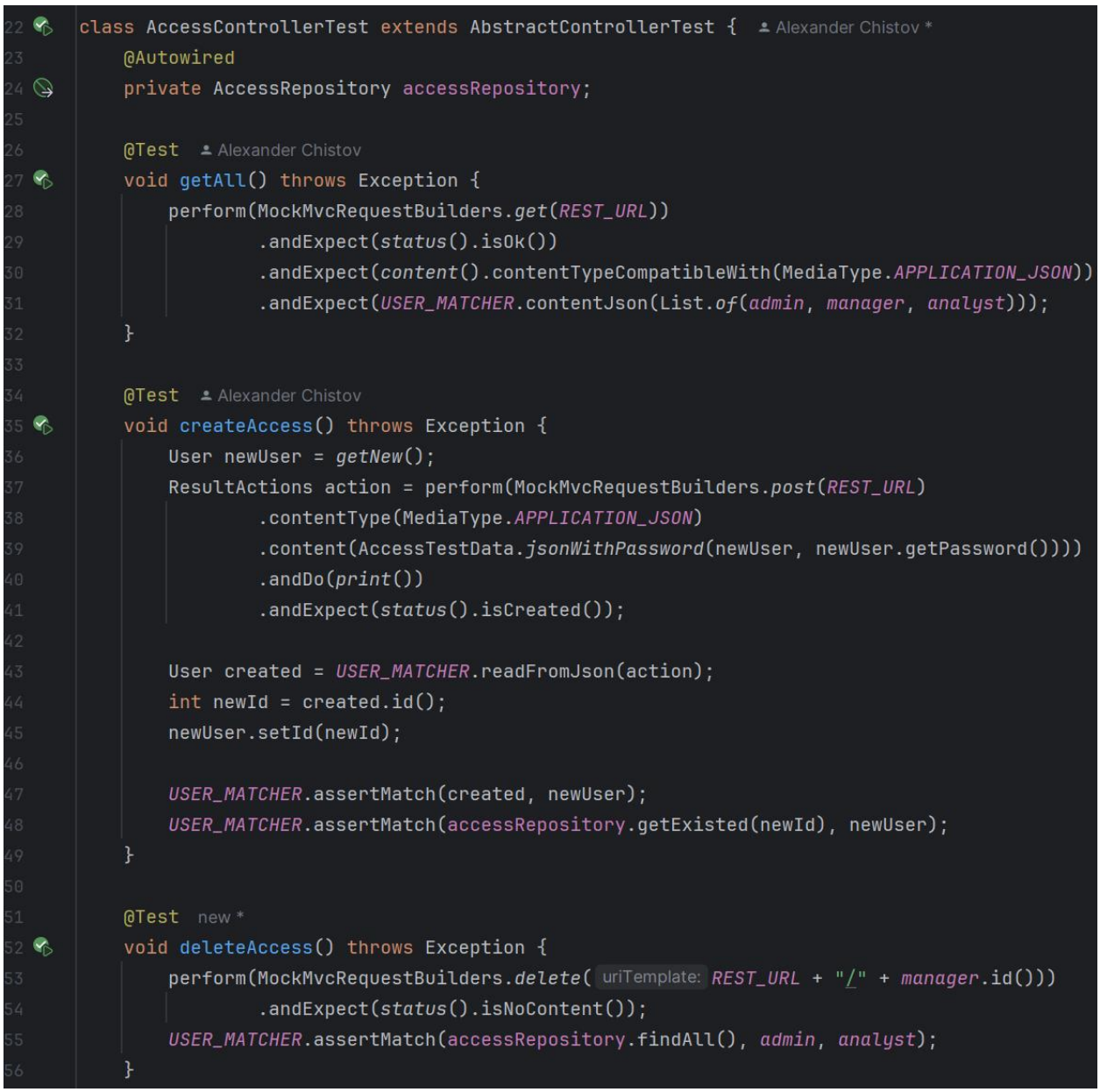


Рисунок 38 – Тестовые данные для тестирования класса AccessController

На рисунке 39 представлен модульный тест, написанный для тестирования REST API класса AccessController. Этот контроллер обрабатывает запросы по указанному адресу. Так же в коде есть модель User и сервис, которые работают с данными. Admin, manager и analyst – это заранее подготовленные моковые данные.



```
class AccessControllerTest extends AbstractControllerTest {  Alexander Chistov *
    @Autowired
    private AccessRepository accessRepository;

    @Test  Alexander Chistov
    void getAll() throws Exception {
        perform(MockMvcRequestBuilders.get(REST_URL))
                .andExpect(status().isOk())
                .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON))
                .andExpect(USER_MATCHER.contentJson(List.of(admin, manager, analyst)));
    }

    @Test  Alexander Chistov
    void createAccess() throws Exception {
        User newUser = getNew();
        ResultActions action = perform(MockMvcRequestBuilders.post(REST_URL)
                .contentType(MediaType.APPLICATION_JSON)
                .content(AccessTestData.jsonWithPassword(newUser, newUser.getPassword())))
                .andDo(print())
                .andExpect(status().isCreated());

        User created = USER_MATCHER.readFromJson(action);
        int newId = created.id();
        newUser.setId(newId);

        USER_MATCHER.assertMatch(created, newUser);
        USER_MATCHER.assertMatch(accessRepository.getExisted(newId), newUser);
    }

    @Test  new *
    void deleteAccess() throws Exception {
        perform(MockMvcRequestBuilders.delete( uriTemplate: REST_URL + "/" + manager.id()))
                .andExpect(status().isNoContent());
        USER_MATCHER.assertMatch(accessRepository.findAll(), admin, analyst);
    }
```

Рисунок 39 – Модульный тест на методы класса AccessController

В данном тесте мы проверяем методы контроллера: getAll, create и delete, а также работу базы данных Accesses. MockMvc – это класс из Spring Framework, который используется для тестирования контроллеров Spring MVC без запуска полноценного веб-сервера. Он предоставляет удобный способ выполнения HTTP-запросов и проверки результатов. Метод perform, предоставляемый Spring MockMvc, выполняет HTTP-запрос. Класс MockMvcRequestBuilders – предоставляет статические методы для создания

HTTP-запросов (например, get, post, put, delete). REST_URL – это константа (переменная), содержащая URL-адрес REST API, который мы тестируем. Метод andExpect используется для проверки ожидаемого результата после выполнения HTTP-запроса. Метод content позволяет проверять содержимое ответа HTTP. USER_MATCHER – это пользовательский класс, он имеет методы для сравнения JSON-ответа с ожидаемым списком объектов.

На рисунке 40 рассмотрим тесты, которые проверяют некорректные данные, введенные пользователем в веб-интерфейсе, а также id пользователя, которого не существует. Результат тестирования представлен на рисунке 41.

```
@Test  Alexander Chistov
void createAccessInvalid() throws Exception {
    User newUser = new User( id: null,  fullName: null,  email: "email@mail.ru",  password: "pass", Role.ANALYST);
    perform(MockMvcRequestBuilders.post(REST_URL)
    .contentType(MediaType.APPLICATION_JSON)
            .content(JsonUtil.writeValue(newUser)))
            .andExpect(status().isUnprocessableEntity());
}


@Test  Alexander Chistov
void deleteAccessInvalid() throws Exception {
    perform(MockMvcRequestBuilders.delete( uriTemplate: REST_URL + "/100"))
            .andExpect(status().isNotFound());
}
```

Рисунок 40 – Модульный тест, покрывающих негативные сценарии методов

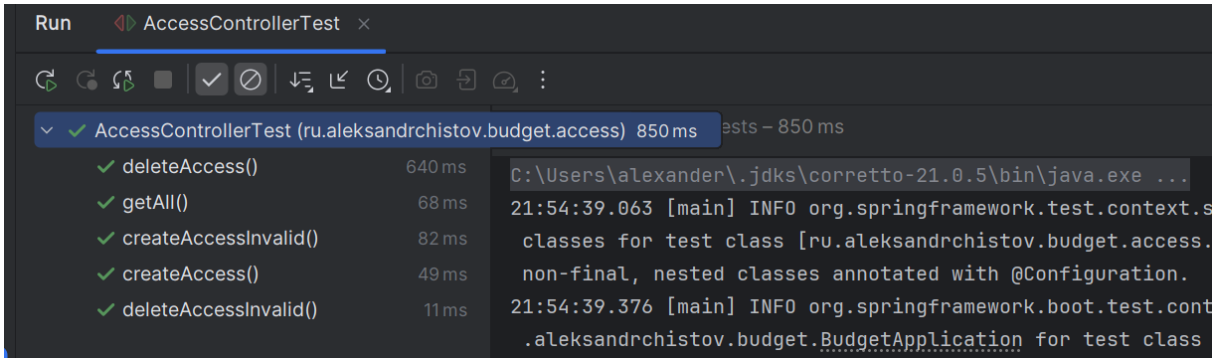


Рисунок 41 – Результат тестирования методов класса AccessController

Тестирование негативных сценариев позволяет найти слабые места в коде, которые могут привести к ошибкам в реальных условиях, и решить эти проблемы на ранних стадиях. Это предотвращает проблемы с производительностью, стабильностью и безопасность. Например, если приложение может обработать некорректный ввод данных, это повышает надежность приложения и предотвращает внезапные сбои.

Следующим шагом станет тестирование пользовательского интерфейса (UI). Протестируем работу класса AmountPipe. Пайпы являются мощным инструментом в Angular, который позволяет преобразовывать и форматировать данные, отображаемые в шаблонах, делая код чище и более читаемым. AmountPipe разделяет числа на разряды, а если символов больше 6, он делит сумму на 1000, преобразуя сумму к строке с плавающей запятой и буквой К (рисунок 42). Результат тестирования представлен на рисунке 43.

```
describe('AmountPipe', () => {  Alexander Chistov

  it('create an instance', () => {
    const pipe = new AmountPipe();
    expect(pipe).toBeTruthy();
  });

  it('Should return a string divided into digits when chars are less than 7', () => {
    const pipe = new AmountPipe();
    const initial = 607854;
    const expected = '607 854';

    const result : unknown  = pipe.transform(initial);

    expect(result).toBe(expected);
  });

  it('Should return a floating-point string with the letter K when chars are more than 7', () => {
    const pipe = new AmountPipe();
    const initial = 1607854;
    const initial2 = 134607854;
    const initial3 = 2134607854;
    const expected = '1 607,9 K';
    const expected2 = '134 607,9 K';
    const expected3= '2 134 607,9 K';

    const result : unknown  = pipe.transform(initial);
    const result2 : unknown  = pipe.transform(initial2);
    const result3 : unknown  = pipe.transform(initial3);

    expect(result).toBe(expected);
    expect(result2).toBe(expected2);
    expect(result3).toBe(expected3);
  });

});
```

Рисунок 42 – Юнит-тесты на проверку класса AmountPipe

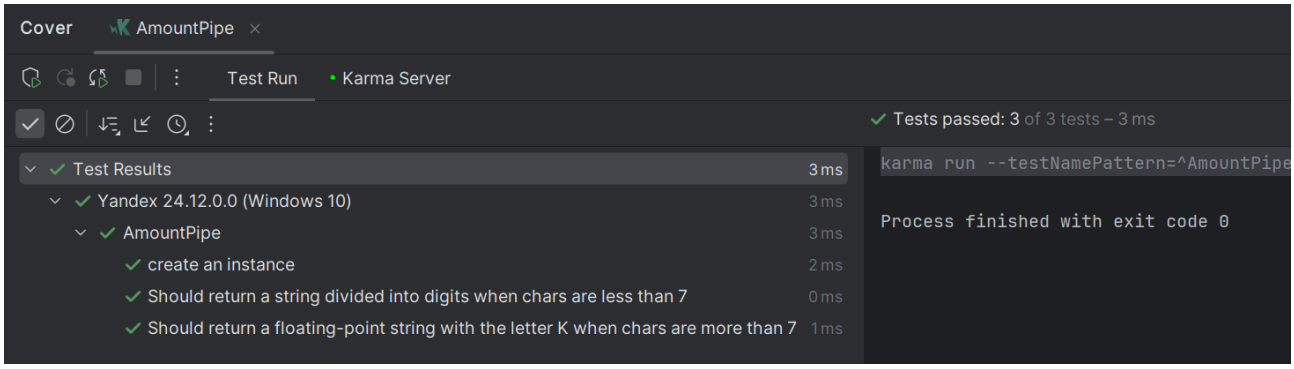


Рисунок 43 – Результат запуска тестов класса AmountPipe

Вывод по разделу 3

В третьем разделе ВКР был осуществлен переход к практической реализации проекта. Разработана диаграмма компонентов, показывающая, как компоненты соединяются вместе, образуя более крупные компоненты и программные системы. Выбраны оптимальные средства реализации веб-приложения, обеспечивающие эффективность и удобство разработки. Создана база данных с необходимыми таблицами для хранения информации. Реализована серверная логика на языке Java, отвечающая за обработку данных и взаимодействие с базой данных. Реализован веб-интерфейс на языке TypeScript, обеспечивающий интерактивное взаимодействие с пользователем. Для обеспечения качества и надежности разработанного решения были написаны модульные тесты, подтверждающие корректность работы основных компонентов веб-приложения.

Заключение

В рамках настоящей выпускной квалификационной работы было успешно разработано веб-приложение учета финансов, предназначенное для оптимизации процессов планирования и анализа бюджетов различных отделов банка на единой платформе. Достигнутая цель подтверждается выполнением всех поставленных задач, что позволило создать функциональное и эффективное решение, отвечающее современным требованиям к информационным системам в финансовой сфере.

В ходе исследования предметной области был проведен тщательный анализ существующих процессов управления бюджетом в банковской организации, выявлены их особенности, недостатки и потребности в автоматизации. На основе полученных данных была выбрана оптимальная системная архитектура, обеспечивающая масштабируемость, надежность и безопасность разработанного веб-приложения. Были определены и формализованы требования к функциональности системы, охватывающие основные аспекты планирования, учета и анализа финансовых данных. Спроектирована диаграмма вариантов использования, наглядно отображающая взаимодействие пользователей с системой. Выбранная методология проектирования позволила эффективно организовать процесс разработки и обеспечить соответствие разрабатываемого веб-приложения требованиям заказчика.

В процессе технической реализации был разработан ряд ключевых компонентов. Была спроектирована и создана база данных для хранения финансовых данных, обеспечивающая целостность, доступность и защиту информации. Выбраны оптимальные средства реализации программного приложения, обеспечивающие высокую производительность и удобство разработки. Разработан и реализован интуитивно понятный интерфейс веб-приложения, обеспечивающий удобный доступ к функциям планирования, учета и анализа. Разработана серверная часть веб-приложения,

обеспечивающая обработку данных, бизнес-логику и взаимодействие с базой данных. Для обеспечения качества и надежности работы системы было проведено тщательное тестирование базы данных и прикладных программ, выявлены и устранены все обнаруженные ошибки и недостатки.

Разработанное веб-приложение обладает значительным потенциалом для повышения эффективности управления бюджетом в банке. Его внедрение позволит автоматизировать рутинные операции, сократить время на подготовку отчетности, улучшить контроль за финансовыми потоками и принимать более обоснованные управленческие решения. Созданная единая платформа для планирования и анализа бюджетов обеспечит прозрачность и согласованность данных, что способствует повышению эффективности работы всех подразделений банка.

Таким образом, цели и задачи, поставленные в начале данной выпускной квалификационной работы, были успешно достигнуты. Результатом работы является готовое к дальнейшему внедрению веб-приложение учета финансов, способное оказать положительное влияние на процессы управления бюджетом в банковской организации. Дальнейшие исследования могут быть направлены на расширение функциональности веб-приложения, интеграцию с другими банковскими системами и разработку аналитических инструментов для поддержки принятия стратегических финансовых решений.

Список используемой литературы

1. Аникеев Д. В. Архитектура информационных систем : учебное пособие. Рязань : Рязанский государственный радиотехнический университет, 2022. 72 с.
2. Бабич А. В. Введение в UML : учебное пособие. 4-е изд. Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2022. 198 с.
3. Вышли майские рейтинги самых популярных языков программирования от Tiobe и PYPL [Электронный ресурс] : URL: https://habr.com/ru/news/812749/ (дата обращения: 30.12.2024).
4. Гвозденко Н. П., Суслова С. А. Разработка блок-схем алгоритмов : учебное пособие. Липецк : Липецкий государственный технический университет, ЭБС АСВ, 2021. 59 с.
5. Герштейн Ю. М. Информационные технологии моделирования бизнес-процессов : конспект лекций. Москва : Российский университет транспорта (МИИТ), 2020. 116 с.
6. Как работают базы данных в IT: разбор на примерах [Электронный ресурс] URL: https://practicum.yandex.ru/blog/chto-takoe-bazy-dannyh/ (дата обращения: 26.12.2024).
7. Леоненков А. В. Объектно-ориентированный анализ и проектирование с использованием UML и IBM Rational Rose : учебное пособие. 3-е изд. Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. 317 с.
8. Сравнение программ ведения управленческого учета [Электронный ресурс] URL: https://www.itan.ru/resheniya-itan/sravnenie-produktov-po-upravlencheskomu-uchetu/ (дата обращения: 30.10.2024).
9. Сунгатуллина А. Т. Системный анализ и функциональное моделирование бизнес-процессов на основе структурного подхода : учебно-

методическое пособие по дисциплине «Моделирование бизнес -процессов». Москва : Российский университет транспорта (МИИТ), 2021. 115 с.

10. Сущности и связи: как и для чего системные аналитики создают ER-диаграммы [Электронный ресурс] URL: https://practicum.yandex.ru/blog/chto-takoe-er-diagramma/ (дата обращения: 25.12.2024).

11. Сычев А. В. Теория и практика разработки современных клиентских веб-приложений : учебное пособие. 4-е изд. Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2025. 482 с.

12. Что такое коммерческий банк [Электронный ресурс] URL: https://journal.sovcombank.ru/glossarii/chto-takoe-kommercheskii-bank (дата обращения: 25.10.2024).

13. Шитов В. Н. Банковское дело : учебное пособие. Ульяновск : Ульяновский государственный технический университет, 2022. 126 с.

14. Шуваев А. В. Методология и технология проектирования информационных систем : учебное пособие для магистрантов направления подготовки 09.04.03 «Прикладная информатика». Ставрополь : Ветеран, 2021. 90 с.

15. Юнит-тесты: что это такое, зачем они нужны и как их проводят [Электронный ресурс] URL: https://practicum.yandex.ru/blog/chto-takoe-modulnoe-yunit-testirovanie/ (дата обращения: 02.01.2025).

16. How To Make An Interface Structure Diagram In Visio [Электронный ресурс] : Process Street. URL: https://www.process.st/how-to/make-an-interface-structure-diagram-in-visio/ (дата обращения: 02.01.2025).

17. Postgres vs. MySQL: a Complete Comparison in 2024 [Электронный ресурс] URL: https://www.bytebase.com/blog/postgres-vs-mysql/ (дата обращения: 26.12.2024).

18. Quick Guide on Loading Initial Data with Spring Boot [Электронный ресурс] URL: https://www.baeldung.com/spring-boot-data-sql-and-schema-sql (дата обращения: 26.12.2024).

19. Top 10 JavaScript Frameworks to Use in 2025 [Электронный ресурс] URL: https://www.geeksforgeeks.org/top-javascript-frameworks/ (дата обращения: 02.01.2025).

20. Web [Электронный ресурс] URL: https://docs.spring.io/spring-boot/reference/web/index.html (дата обращения: 30.12.2024).